Technology
Arts Sciences
TH Köln
Campus Gummersbach

Bachelorarbeit

Entwickeln und Veröffentlichen von Web-Apps mit Hilfe von Microsoft Visual Studio auf Basis von ASP.NET 5 MVC 6 RC1.

Verfasser:

John Marc Sangermann

Matrikelnummer:

11070719

Studiengang:

Medieninformatik

Betreuer:

1. Prof. Dipl.-Des. Christian Noss

2. B.Sc. Christopher Rasche

Gummersbach, im August 2016

1. Inhaltsverzeichnis

2. Abbildungsverzeichnis

3. Einleitung

3.1. Ausgangslage

ASP.NET stellt Frameworks für die Erstellung von dynamischen Web-Apps zur Verfügung. ASP.NET steht für Active Server Pages und ist Bestandteil des .NET-Frameworks, daher die Namensgebung. Die erste Vorabversion wurde vor mehr als 14 Jahren veröffentlicht.[1]

ASP.NET 5 MVC 6 RC1 ist eine bedeutende Neugestaltung von ASP.NET, welche auch ASP.NET Core 1.0 Preview 1 genannt wird. ASP.NET 5 MVC 6 RC1 ist ein optimiertes Entwicklungsframework, mit dem Web-Apps in der eigenen Umgebung oder in der Cloud veröffentlicht werden können. Mit der Microsoft Azure Cloud Plattform lassen sich Webseiten direkt in einer Cloud-Umgebung hosten bzw. veröffentlichen.

In der Bachelorarbeit wird das Thema "Entwickeln und Veröffentlichen von Web-Apps mithilfe von Microsoft Visual Studio auf Basis von ASP.Net 5 MVC 6 RC1" behandelt. Visual Studio wird in der Version Enterprise 2015 genutzt. Als Webseitentechnologie kommt die aktuelle Version des ASP.NET MVC Frameworks zum Einsatz. Die aktuelle Version lautet ASP.NET 5 MVC 6 RC1. Wie die Bezeichnung der Version vermuten lässt, basiert das Framework auf dem Model-View-Controller Paradigma und befindet sich im Status Release Candidate 1. Bei einem Release Candidate handelt es sich um ein Entwicklungsstadium vor der offiziellen Veröffentlichung. Die Syntax mit Namen Razor kommt bei ASP.NET MVC zum Einsatz.

3.2. Ziel der Arbeit

Ziel dieser Arbeit ist es einen Entwicklungs- und Veröffentlichungsprozess einer ASP.NET 5 MVC 6 RC1 Web-App beschrieben zu haben. Mit dieser Arbeit soll es möglich sein, dass sich sowohl professionelle Entwickler, als auch Programmieranfänger in ASP.NET 5 MVC 6 RC1 einarbeiten und ihre eigenen Web-Anwendungen erstellen können.

3.3. Vorgehen

Zu Beginn der Arbeit wird auf das eingesetzte Framework ASP.NET eingegangen. Das Model-View-Controller Paradigma und seine Funktionsweise wird beschrieben. Die Entwicklungsumgebung Microsoft Visual Studio in Version 2015 Enterprise wird ebenfalls kurz erläutert. Anschließend wird eine Einleitung der Programmiersprache C# gegeben. Es werden Beispielanwendungen programmiert, die als weitere Erklärungen dienen. Die Razor-Syntax wird erläutert. Eine Beispielanwendung wird zum Ende auf einem Webserver in der Microsoft Azure Cloud gehostet und veröffentlicht. Zum Ende der Arbeit wird ein Fazit formuliert, welches den Grad der Zielerreichung erläutert und einen Ausblick gibt. Offene Fragen werden ergänzend dargelegt.

---

[1] Vgl. (Microsoft Developer Network, 2016)

3.4. Erwartete Ergebnisse

Am Ende der Bachelorarbeit ist eine ASP.NET 5 MVC 6 RC1 Web-Anwendung mithilfe von Microsoft Visual Studio Enterprise 2015 erstellt, die auf der Microsoft Azure Cloudplattform gehostet und lauffähig ist. Der Prozess des Entwickelns und der Veröffentlichung einer Web-App ist somit abgeschlossen.

3.5. Zielgruppe

Die Arbeit richtet sich an all diejenigen Personen, die im Umfeld der Webentwicklung aktiv sind. Dazu gehören Webentwickler, Programmierer von Hostingplattformen, wie Webserver, aber genauso die Anbieter von Webpaketen, Hosting- und Domaindiensten. Ergänzend sind die Personen als Zielgruppe zu nennen, die mit Cloud-Diensten, wie der Azure Plattform bereits arbeiten und entwickeln. Im Allgemeinen ist diese Arbeit an Menschen gerichtet, die sich mit dem Thema Web-Apps, Hosting und Cloud auseinandersetzen oder generelles Interesse an diesen Themen haben.

3.6. Eigene Motivation

Der Verfasser der Arbeit ist bereits neben dem Studium hauptberuflich im Microsoft-Technologie-Umfeld beschäftigt und dort als Softwareentwickler tätig. Neue Projekte für Kunden werden, bis auf wenigen Ausnahmen, nur noch als Web-Apps umgesetzt. Ebenso werden bestehende Softwarelösungen zu Webanwendungen migriert. Durch den vermehrten Einsatz von Web-Apps und Cloud-Diensten, ist es auch für Entwickler notwendig, sich auf neue Gebiete bzw. auf neue Technologien zu konzentrieren. Da das Arbeitsumfeld des Authors auf Microsoft-Technologien ausgelegt ist und Web-Apps als Softwarelösung hauptsächlich bei dessen Tätigkeit zum Einsatz kommen, fiel die Entscheidung auf dieses Thema. Dazu kam, dass die Microsoft Azure Cloud Plattform stetig wächst. Ebenfalls ist zu erwähnen, dass die Entwicklung von Web-Apps und der Begriff Cloud an sich und die damit verbundenen Technologien, zu den persönlichen Interessen des Autors zählen.

4. ASP.NET 5 MVC 6 RC1

In diesem Kapitel wird eine Einführung in das Framework ASP.NET gegeben. Darauffolgend wird das MVC-Architekturschema beschrieben. Anschließend wird die Verknüpfung des ASP.NET-Frameworks und des MVC-Architekturschemas hergestellt.

4.1. Einführung in ASP.NET

ASP.NET bietet drei Frameworks für die Erstellung von dynamischen Web-Apps. Diese lautet ASP.NET Websites, ASP.NET Web Forms und ASP.NET MVC. ASP.NET Websites konzentriert sich auf HTML-Seiten und bietet eine einfache und leichte Syntax, um dynamischen und auch serverseitigen Code auszuführen. Ebenfalls ist der Zugriff auf Datenbanken möglich. ASP.NET Web Forms bietet traditionelle Fenstertyp-Steuerelemente, wie Schaltflächen, Listen, Buttons, etc. an. Diese Steuerelemente kennt man von normalen Desktopanwendungen und werden in den Webbrowser portiert. ASP.NET MVC implementiert das Model-View-Controller-Paradigma für das ASP.NET-Framework. Die Betonung liegt auf der Trennung der Bereiche Geschäftslogik (Model), Präsentationslogik (View) und Eingabelogik (Controller).

4.2. Das MVC-Architekturschema

MVC ist die Abkürzung für Model-View-Controller. Das Model, die View und der Controller sind die drei Hauptkomponenten des MVC-Architekturschemas. Die Geschäftslogik ist Teil des Models. Die Präsentationslogik ist Teil der View und die Eingabelogik ist Bestandteil des Controllers. Mithilfe dieses Architekturschemas lässt sich die Anwendung in Teilbereiche zerlegen und die Komplexität verringern. Ebenfalls können diese drei Komponenten unabhängig voneinander behandelt und entwickelt werden. Ein Entwickler könnte sich um die Views kümmern, ein Weiterer programmiert den Controller und ein dritter Programmierer entwickelt das Model. Folgend werden nun die drei Komponenten im Detail erläutert. [2]

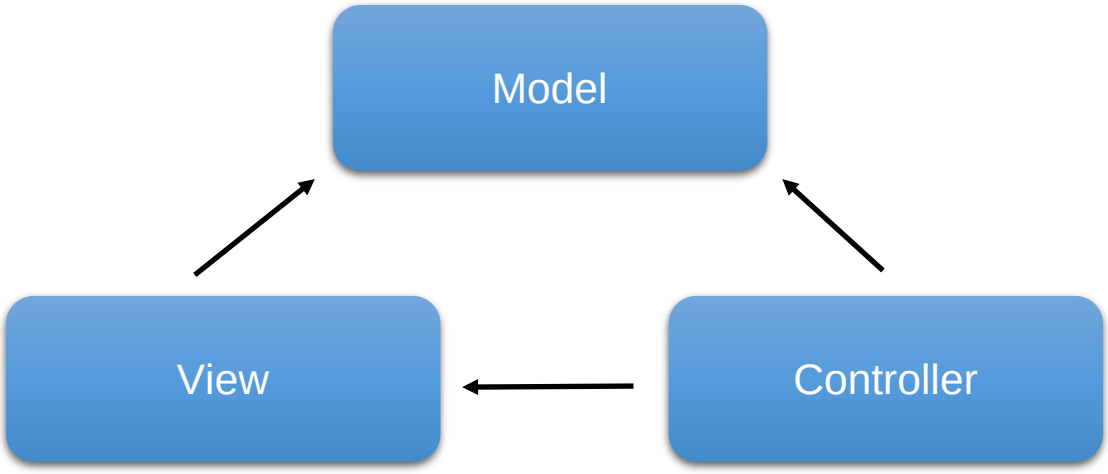


Abbildung 4.2.1 Model-View-Controller-Aufbau

[2] Vgl. (Microsoft Developer Network, 2010), ASP.NET MVC Overview

4.2.1.Model

Das Model beinhaltet die Datenstruktur der Anwendung. Beispielsweise eine Datenbank. Dabei kann mithilfe von vier Datenbankoperationen ein Zugriff auf bzw. eine Manipulation der gespeicherten Daten erfolgen. Diese vier Datenbankoperationen werden als CRUD-Akronym zusammengefasst:

1. Create Datensatz werden in der Datenstruktur erstellt
2. Read Angelegte Datensätze werden abgerufen
3. Update Vorhandene Datensätze werden aktualisiert
4. Delete Datensätze werden gelöscht

Bei nicht komplexen Anwendungen ist das Model häufig nur konzeptionell und nicht physisch von den anderen Komponenten getrennt. Greift eine Anwendung beispielsweise auf zwei Zahlen-Variablen zurück, die für eine Addition benötigt werden, verfügt die Anwendung über keine physische Datenstruktur.

4.2.2.View

Die View beinhaltet die Präsentationslogik. Die Benutzeroberflächen werden abgebildet und in der Regel aus dem Model erzeugt. Beispielsweise können dem Benutzer Datensätze aus einer Datenbank angezeigt werden. Über die Views werden ebenso die Benutzerinteraktionen abgebildet. Dabei ist die View nicht für die direkte Manipulation der Datensätze des Models zuständig, sondern der Controller. Hierauf wird im nächsten Abschnitt eingegangen.

4.2.3.Controller

Ein Controller tritt als Vermittler zwischen der View und dem Model auf und übernimmt das Behandeln und das Reagieren auf Benutzerinteraktion. Dabei wählt der Controller die anzuzeigende View aus. Zu jeder View existiert ein eigener Controller. Als Beispiel trägt der Benutzer in einer View Benutzername und Kennwort ein. Der Controller leitet die beiden Benutzerinformationen an das Model weiter und enthält ggf. einen Wert zurück, ob der Anmeldevorgang erfolgreich war oder nicht.

4.3. MVC im ASP.NET-Framework

Im ASP.NET-Framework spiegelt sich das MVC-Paradigma direkt wider. Trennung der drei Hauptkomponenten Geschäftslogik, Präsentationslogik und Eingabelogik wird durchgeführt. Ergänzend ist zu erwähnen, dass die Schnittstellen der drei Hauptkomponenten über Pseudoobjekten abbilden werden können. Pseudoobjekte sind simulierte Objekte, die das Verhalten der tatsächlichen Objekte der Applikation imitieren. Dies wird für Komponententests zur Verfügung gestellt. Es ist dadurch möglich die drei Komponenten komplett entkoppelt voneinander zu behandeln.[3]

[3] Vgl. (Microsoft Developer Network, 2010), ASP.NET MVC Overview

5. Microsoft Visual Studio Enterprise 2015

In diesem Kapitel wird auf die eingesetzte Entwicklungsumgebung eingegangen und kurz Hintergrundinformationen dazu gegeben.

5.1. Was ist Visual Studio?

Bei Microsoft Visual Studio Enterprise 2015 handelt es sich um die aktuelle Entwicklungsumgebung der Firma Microsoft. Eine Entwicklungsumgebung ist eine Softwarelösung, mit der sich selbst Softwarelösungen entwickeln und programmieren lassen. Visual Studio Enterprise 2015 unterstützt zudem plattformübergreifende Entwicklung von Applikationen. Apps können für iOS, Android, Linux und Windows erstellt werden.[4]

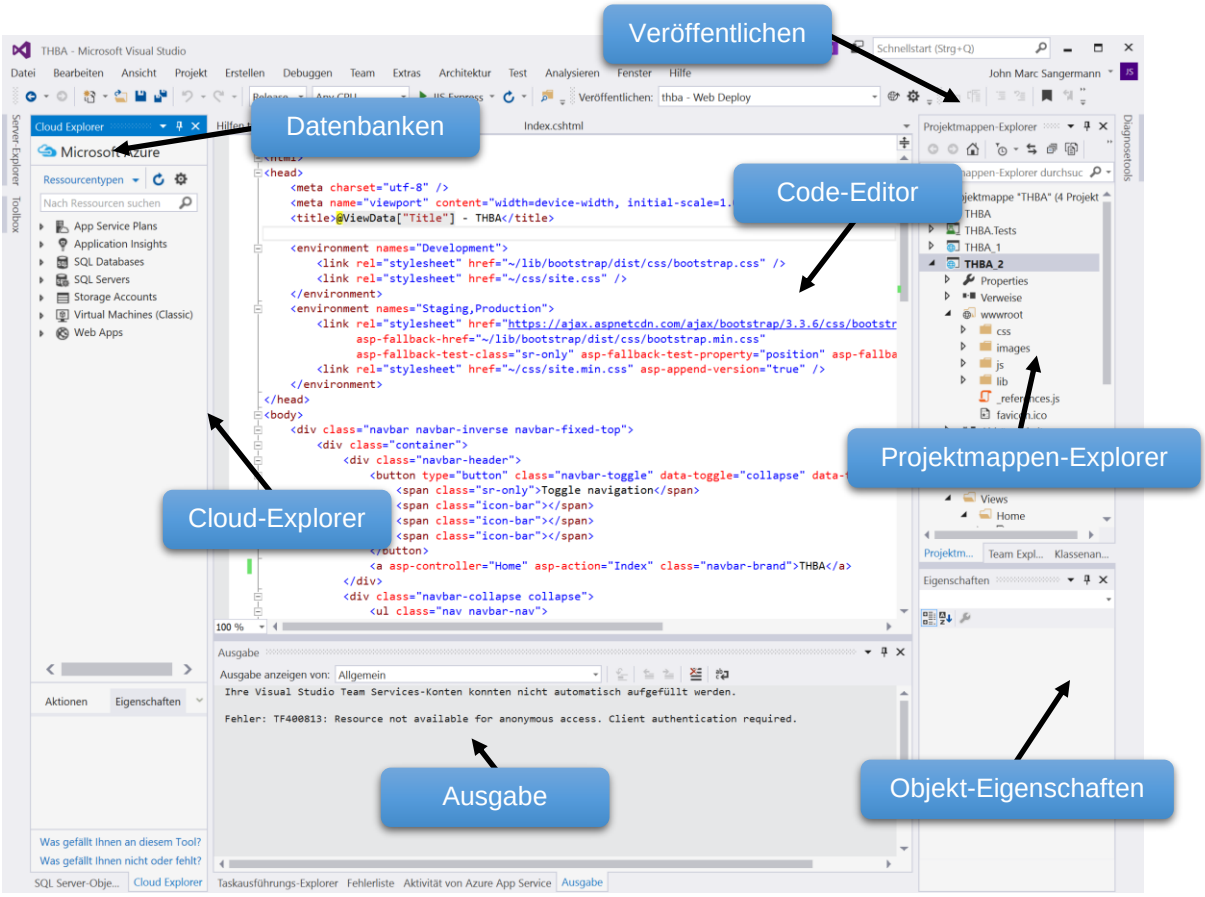


Abbildung 5.1.1 Arbeitsbereich von Visual Studio Enterprise 2015

5.2. Anpassungen in Visual Studio

Visual Studio Enterprise 2015 beinhaltet jegliche Tools zum Erstellen von Softwarelösungen vom Entwurf, über das Codieren, Testen, Debuggen und das Analysieren der

[4] Vgl. (Microsoft Developer Network, 2015), IDE Basics

Codequalität, bis hin zur Bereitstellung der Software in der finalen Version. Visual Studio Enterprise 2015 unterstützt standardmäßig die Programmiersprache C#, C und C++, JavaScript, F# und Visual Basic. Visual Studio Enterprise 2015 lässt sich mit Drittanbieteranwendungen einsetzen. Anpassungen an die Benutzeranforderungen lassen sich Visual Studio Enterprise 2015 durchführen. Für jeden Entwicklungsstil gibt es unterschiedliche Layouts. Frontend und Backend lassen sich separat oder kombiniert anzeigen. Ferner gibt es eine Live-Vorschau des Frontends. Fenster der Oberfläche lassen sich jederzeit ausblenden, andocken oder die Verankerung aufheben. Die Entwicklungsumgebung unterstützt mehrere Monitore. Zudem gibt es eine von Microsoft geführte Liste von Tastenkombinationen, um das Arbeiten mit Visual Studio Enterprise 2015 zu optimieren und zu beschleunigen. [5]

5.3. Erweiterungen in Visual Studio

Über die Visual Studio Gallery lassen sich Erweiterungen nachinstallieren. Eine Standarderweiterung ist der NuGet Package Manager, mit dem sich aus einem Repository einige externe Tools direkt installieren lassen. Als Beispiel sei hier einmal die freie Javascript- Bibliothek jQuery genannt. Diese wird dann passend zum Projekt in dem Script-Ordner abgelegt, ohne die Bibliothek manuell von einer Webseite herunterladen zu müssen.

6. Programmiersprache C#

In dieser Arbeit wird die Programmiersprache C# zum Einsatz kommen. In diesem Kapitel wird oberflächlich auf die eingesetzte Programmiersprache C# eingegangen.

6.1. Was ist C#

C# ist eine Programmiersprache, die zum Erstellen einer großen Bandbreite von Anwendungen, die auf .NET-Framework aufsetzen, entwickelt wurde. C# ist einfach strukturiert, leistungsfähig, typsicher und objektorientiert. Die zahlreichen Neuerungen in C# ermöglichen eine schnelle Anwendungsentwicklung, bei der die Ausdruckskraft und Eleganz von Sprachen im C-Format erhalten bleibt.

Visual C# stellt eine Microsoft-Implementierung der Programmiersprache C# dar. In Visual Studio Enterprise 2015 wird Visual C# durch einen umfassenden Code-Editor, einen Compiler, Projektvorlagen, Designer, Code-Assistenten, einen leistungsfähigen und einfach zu bedienenden Debugger sowie weitere Tools unterstützt. Die .NET-Framework-Klassenbibliothek ermöglicht den Zugriff auf viele Betriebssystemdienste und weitere nützliche, ausgereifte Klassen, mit denen der Entwicklungsprozess deutlich beschleunigt wird.[6]

[5] Vgl. (Microsoft Developer Network, 2015), Personalizing the IDE
[6] Vgl. (Microsoft Developer Network, 2015), C#

7. Razor

Dieses Kapitel behandelt die Razor-Syntax. Es wird gezeigt, um was es sich bei dieser Syntax im Allgemeinen handelt. Der Zusammenhang von Razor und ASP.NET wird aufgezeigt. Ebenfalls wird die Syntax anhand von Beispielen erläutert.

7.1. Was ist Razor

Razor ist keine Programmiersprache, sondern eine Markup-Sprache. Mit Razor ist es möglich eingebetteten serverseitigen Code direkt in eine Webseite einzubinden. Es gibt somit zwei Arten von Inhalt. Der clientseitige Inhalt besteht aus dem HTML-Markup, Styleinformationen wie CSS, Clientskripte wie Javascript und Klartext. Der serverseitige Code wird auf dem Server direkt ausgeführt. Dieser Code wird vor dem Clientseitigen Code kompiliert, d.h. bevor die Seite an den Browser geschickt wird. Durch diese Trennung ist es möglich komplexe Aufgaben, wie Datenbankanbindungen über ein Model direkt durchzuführen ohne vom Client Daten an den Server senden zu müssen. Durch die serverseitige Verarbeitung lassen sich Inhalte dynamisch generieren und an den Client senden. Daraus ergibt sich, dass eine Webseite aus statischen, clientseitigem Code bzw. Markup und dynamischen, serverseitigen Code besteht. [7]

7.2. ASP.NET und Razor

ASP.NET Webseiten, die die Razor-Syntax integrieren, haben entweder die Datei-Endung `cshtml` oder `vbhtml`. Bei `cshtml`, welches in dieser Arbeit benutzt wird, kommt die Programmiersprache C# zum Einsatz. Bei `vbhtml` wird Visual Basic benutzt. Der Webserver erkennt die Dateiendungen, durchläuft den Code und führt den serverseitigen, mit der Razor-Syntax deklarierten Programmcode, aus. Die Seite wird anschließend an den Client, beispielsweise den Browser, geschickt.

Die Razor-Syntax basiert auf einer XML-basierten Technologie von Microsoft, die Inhalt des .NET-Frameworks ist. ASP.NET ist ebenfalls Bestandteil dieses Frameworks. Razor wird jedoch nur in Kombination mit ASP.NET eingesetzt.

[7] Vgl. (Microsoft ASP.NET, 2014), The Razor Syntax, Server Code, and ASP.NET

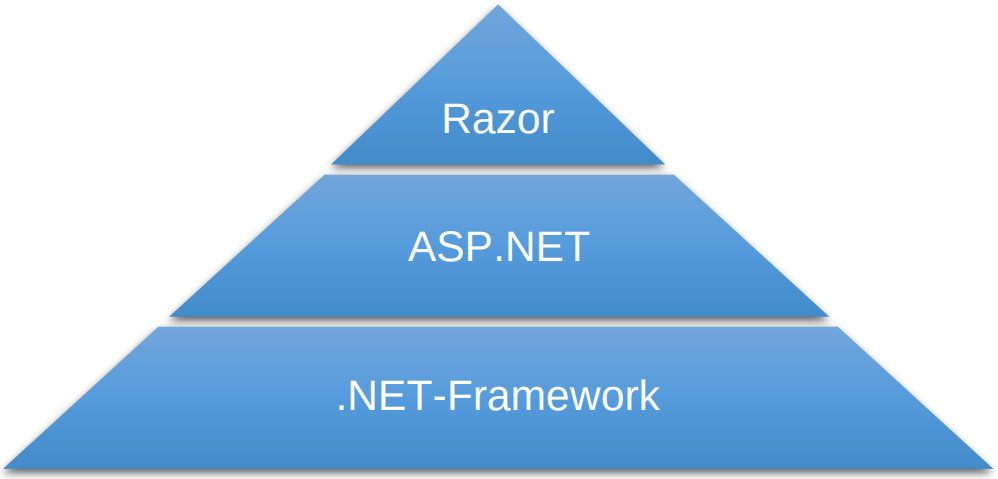


Abbildung 7.2.1 Zusammenhang von Razor und ASP.NET

7.3. Die Syntax von Razor

Razor-Codeblöcke werden allgemein immer mit dem @-Operator eingeleitet. Die Syntax für Razor lautet:

```
// Razor-Syntax einzeiliger Code
@{ var zahl = 42; }

// Razor-Syntax mehrzeiliger Code
@{
    var nochEineZahl = 56;
    var zeichenfolge = "Hello World";
}
```

Quellcode 7.3.1 Syntax für Razor

Auf den einleitenden @-Operator folgt eine öffnende, geschweifte Klammer. Die innere Code-Anweisung wird mit einem Semikolon beendet. Die schließende, geschweifte Klammer beendet den Razor-Code-Block. Einzeilige Razor-Codeblöcke können aber auch den Aufbau @( ) oder @: haben. Da eine ASP.NET-Webseite erst serverseitig durchlaufen wird, muss gekennzeichnet sein, ob es sich um server- oder clientseitigen Code handelt. Dies kann auf mehrere Arten geschehen.

Umschließen Sie den Code in einem HTML-Element:

```
@{
    // HTML-Elemente deklarieren clientseitigen Code
    <p>Hello World</p>
}
```

Quellcode 7.3.2 HTML-Elemente deklarieren clientseitigen Code

Ein HTML-Element kann aber auch zusätzlich serverseitigen Code enthalten. ASP.NET erkennt diesen automatisch.

```
@{
    // HTML-Elemente deklarieren clientseitigen Code
    // @-Zeichen deklariert servseitigen Code im HTML-Element
    <p>Hello World um @DateTime.Now</p>
}
```

Quellcode 7.3.3 Servercode im HTML-Element

Eine andere Möglichkeit für die Trennung besteht darin, dass der @-Operator oder das Pseudo-Element `<text>` verwendet wird. `@:` gibt grundsätzlich eine einzelne Textzeile aus.

```
@{
    // Einzeiliger Code mit @:
    @: Hello World um @DateTime.Now
}
```

Quellcode 7.3.4 Einzeiliger Code mit @-Operator

Mit dem Pseudo-Element `<text>` kann mehrzeiliger Code deklariert werden.

```
@{
    // Mehrzeiliger Code mit <text>
    <text>
        Heute ist der @DateTime.Now.ToShortDateString()
        <br />
        und es ist @DateTime.Now.ToShortTimeString() Uhr
    </text>
}
```

Quellcode 7.3.5 Mehrzeiliger Code mit `<text>`-Element

Mehrzeiliger Code kann ferner über `@:`-Elemente dargestellt werden, die in jede Zeile geschrieben werden.

```
@{
    // Mehrzeiliger Code mit @:
    @: Heute ist der @DateTime.Now.ToShortDateString()
    @: <br />
    @: und es ist @DateTime.Now.ToShortTimeString() Uhr
}
```

Quellcode 7.3.6 Mehrzeiliger Code mit `@:`-Elementen

Diese Elemente verwendet ASP.NET um Textinhalte zu identifizieren. Diese Elemente werden niemals gerendert und somit im Seitenquelltext im Browser nicht angezeigt.[8]

7.4. Leerzeichen

Überflüssige Leerzeichen in einer Zeile und außerhalb einer Zeichenfolge haben keinen Einfluss auf den Ausdruck der Zeile. Sie werden somit ignoriert.[9]

```
@{
    // Code mit überflüssigen Leerzeichen
    string    name    =        "Max";
}
```

Quellcode 7.4.1 Serverseitiger Code mit überflüssigen Leerzeichen

7.5. Zeilenumbrüche

Ein Zeilenumbruch in einem Ausdruck hat keinen Einfluss auf die Anweisung. Sie werden, wie die Leerzeichen, ignoriert.

```
@{
    // Code mit überflüssigen Zeilenumbrüchen
    string name =

        "Max";
}
```

Quellcode 7.5.1 Serverseitiger Code mit überflüssigen Zeilenumbrüchen

Zu beachten ist, dass inmitten einer Zeichenfolge keine Zeilenumbrüche erlaubt sind.

```
@{
    // Code mit Zeilenumbrüchen in einer Zeichenfolge
    string name = "Max

        Mustermann";
}
```

Quellcode 7.5.2 Zeilenumbrüche in einer Zeichenfolge

[8] Vgl. (Microsoft ASP.NET, 2014), Combining Text, Markup, and Code in Code Blocks
[9] Vgl. (Microsoft ASP.NET, 2014), Whitespaces

Um dies abbilden zu können, verkettet man die Zeichenfolgen entweder mit dem +-Operator oder mit einem erneutem @-Operator. [10]

```
@{
    // Code mit Zeilenumbrüchen in einer Zeichenfolge
    // mit verkettenden +-Operator
    string name = @"Max +


        Mustermann";

    // Code mit Zeilenumbrüchen in einer Zeichenfolge
    // mit erneutem @-Operator
    name = @"Max


        Mustermann";
}
```

Quellcode 7.5.3 Zeilenübergreifende, verkettete Zeichenfolgen

7.6. Kommentare im Quellcode

Es gibt verschiedene Möglichkeiten Razor-Code und HTML-Markup zu kommentieren. Razor-Kommentare bieten den Vorteil, dass diese zur Laufzeit aufgelöst, aber nicht gerendert werden. Sie sind somit für den Client bzw. Browser unsichtbar. Die Razor-Kommentare bilden die gleiche Funktionalität ab wie herkömmliche Kommentare. Razor Kommentare starten immer mit @* und hören mit *@ auf.

```
@* Dies ist ein einzeiliger Razor-Kommentar *@

@*
    Dies ist ein mehrzeiliger
    Razor-Kommentar
*@
```

Quellcode 7.6.1 Ein- und mehrzeiliger Razor-Kommentar

[10] Vgl. (Microsoft ASP.NET, 2014), Whitespaces

Razor-Kommentare können auch in einem Razor-Codeblock stehen.

```
@{
    @* Einzeiliger Razor-Kommentar innerhalb eines Razor-Codeblocks *@

    @*
        Mehrzeiliger Razor-Kommentar
        innerhalb eines Razor-Codeblocks
    *@
}
```

Quellcode 7.6.2 Razor-Kommentare innerhalb eines Razor-Codeblocks

Grundsätzlich ist es bei ASP.NET möglich HTML-Kommentare, oder auch die Kommentare der verwendeten Programmiersprache, in diesem Fall C#, zu verwenden. Dabei ist zu beachten, dass diese Kommentare innerhalb des Razor-Codeblocks stehen müssen. Ergänzend ist zu erwähnen, dass HTML-Kommentare gerendert werden und somit clientseitig sichtbar sind.[11]

```
@{
    <!-- Dies ist ein einzeiliger HTML-Kommentar -->

    <!--
        Dies ist ein mehrzeiliger
        HTML-Kommentar
    -->
}
```

Quellcode 7.6.3 HTML-Kommentare in einem Razor-Codeblock

```
@{
    //Dies ist ein einzeiliger C#-Kommentar

    /*
        Dies ist ein mehrzeiliger
        C#-Kommentar
    */
}
```

Quellcode 7.6.4 C#-Kommentare in einem Razor-Codeblock

7.7. Variablen

Generell sind Variablen Objekte mit Namen, die einem bestimmten Datentyp angehören. Variablen können aber auch ohne Typ deklariert werden. Bezogen auf ASP.NET können Variablen als Zeichenfolge (`string`), als Ganzzahl (`int`) oder auch als Datumswerte (`DateTime`) vorkommen. Es gibt einige weitere Datentypen, auf die hier nicht weiter eingegangen wird. Wird allerdings kein Typ angegeben oder ist dieser

[11] Vgl. (Microsoft ASP.NET, 2014), Code (and Markup) Comments

noch unbekannt, wird der Datentyp Variant (var) genutzt. Hierbei versucht ASP.NET den Datentyp zur Laufzeit zu ermitteln. Variablendeklarationen müssen im Razor-Codeblock eingebettet werden.[12]

```
@{
    // Variable vom Datentyp Zeichenfolge (string)
    string zeichenfolge = "Hello World";

    // Variable vom Datentyp Datum/Uhrzeit (DateTime)
    DateTime datumUhrzeit = new DateTime(2016, 06, 24, 12, 30, 15);

    // Variable vom Datentyp Ganzzahl (int)
    int ganzzahl = 32;

    // Variablen vom Datentyp Variant (var)
    var variabel1 = "Hello World";
    var variabel2 = new DateTime(2016, 06, 24, 12, 30, 15);
    var variabel3 = 32;
}
```

Quellcode 7.7.1 Variablendeklarationen im Razor-Codeblock

Auf Variablen wird in Razor über den @-Operator zugegriffen.

```
@{
    // Variable vom Datentyp Zeichenfolge (string)
    string zeichenfolge = "Hello World";

    // Variable vom Datentyp Datum/Uhrzeit (DateTime)
    DateTime datumUhrzeit = new DateTime(2016, 06, 24, 12, 30, 15);

    // Variable vom Datentyp Ganzzahl (int)
    int ganzzahl = 32;

    // Variablen vom Datentyp Variant (var)
    var variabel1 = "Hello World";
    var variabel2 = new DateTime(2016, 06, 24, 12, 30, 15);
    var variabel3 = 32;
}

@{
    <p>@zeichenfolge</p>

    <p>
        Heute ist der @datumUhrzeit.ToShortDateString() <br />
        und es ist @datumUhrzeit.ToShortTimeString()
    </p>

    <p>Das Ergebnis lautet: @(ganzzahl + 4) </p>
}
```

Quellcode 7.7.2 Zugriff auf Variablen in Razor

[12] Vgl. (Microsoft ASP.NET, 2014), Variables

7.8. Konvertieren von Datentypen

ASP.NET erkennt grundsätzlich um welche Datentypen es sich bei Variablen handelt. Manchmal ist es notwendig ASP.NET explizit mitzuteilen, wenn ein Datentyp vom Typ Zeichenfolge (`string`) in einen Datentyp vom Typ Ganzzahl (`int`) konvertiert werden soll. Dies geschieht beispielsweise bei Benutzereingaben innerhalb eines Textfeldes. Diese werden immer als Zeichenfolge übergeben. In ASP.NET war es bisher wie folgt:

Methode	Beschreibung	Beispiel

AsInt(), IsInt()	Wandelt eine Zeichenfolge (string) in eine Ganzzahl (int) um.	`var ganzzahl = 0;`
		`var zahlAlsZeichenfolge = "42";`
		`if(zahlAlsZeichenfolge.IsInt() == true){`
		`  ganzzahl = zahlAlsZeichenfolge.AsInt();`
		`}`

AsBool(), IsBool()	Wandelt eine Zeichenfolge (string) in einen Boolean (bool) um.	`var booleanAlsZeichenfolge = "True";`
		`if(booleanAlsZeichenfolge.IsBool() == true){`
		`  var boolean = booleanAlsZeichenfolge.AsBool();`
		`}`

AsFloat(), IsFloat()	Wandelt eine Zeichenfolge (string) in eine Fließkommazahl (float) um.	`var fliesskommazahlAlsZeichenfolge = "42.75";`
		`if(fliesskommazahlAlsZeichenfolge.IsFloat() == true){`
		`  var fliesskommazahl = fleskommazahlAlsZeichenfolge.AsFloat();`
		`}`

AsDecimal(), IsDecimal()	Wandelt eine Zeichenfolge (string) in eine Dezimalkommazahl	`var dezimalkommazahlAlsZeichenfolge = "56,12574";`
		`if(dezimalkommazahlAlsZeichenfolge.IsDecimal() == true){`
		`  var dezimalkommazahl = dezimalkommazahlAlsZeichenfolge.AsDecimal();`
		`}`

AsDateTime(), IsDateTime()	Wandelt eine Zeichenfolge (string) in ein Datumstyp (DateTime) um.	`var datumAlsZeichenfolge = "24.06.2016";`
		`if(datumAlsZeichenfolge.IsDateTime() == true){`
		`  var newDate = myDateString.AsDateTime();`
		`}`

ToString()	Wandelt jeglichen Datentyp in eine Zeichenfolge (string) um.	`double grosseFliesskommazahl = 32.653456;` `string zeichenfolge = grosseFliesskommazahl.ToString();`

Abbildung 7.8.1 Konvertierung in Datentypen bisher in ASP.NET

In ASP.NET 5 MVC 6 RC1 sind die Methoden bisher nicht vorhanden. Ebenfalls gibt es noch keinen Eintrag in der ASP.NET 5 MVC 6 RC1 Referenz. Es wird davon ausgegangen, dass ggf. keine Datenkonvertierung mehr seitens ASP.NET durchgeführt wird, sondern direkt von der Programmiersprache. In der Programmiersprache C# sind die Datentypkonvertierungen über die `Convert`-Klasse oder über die `Parse`-Methoden möglich.[13]

```
@{
    var ganzzahlAlsZeichenfolge = "42";

    // Datentypkonvertierung mit Parse-Methode
    int ganzzahl = int.Parse(ganzzahlAlsZeichenfolge);

    // Datentypkonvertierung mit Convert.Klasse
    int ganzzahl2 = Convert.ToInt32(ganzzahlAlsZeichenfolge);
}
```

Quellcode 7.8.1 Konvertierungen mit `Parse` und `Convert`

[13] Vgl. (Microsoft ASP.NET, 2014), Converting and Testing Data Types

7.9. Operator

Ein Operator gibt an, welche Art von Ausdruck durchzuführen ist. Razor unterstützt die folgenden Operatoren: [14]

Operator	Beschreibung	Beispiel
=	Wertzuweisung bei einer Variablen.	`var zeichenfolge = "Hello World";`
==	Wertvergleich. Rückgabe True, wenn Wert gleich ist oder False, wenn Wert ungleich ist.	`if(wert == 42) { }`
!=	Wertvergleich. Rückgabe False, wenn Wert gleich ist oder True, wenn Wert ungleich ist.	`if(wert != 42) { }`
+	Mathematische Operation. Addition.	`var wert = 32 + 8;`
-	Mathematische Operation. Subtraktion.	`var wert = 32 - 8;`
*	Mathematische Operation. Multiplikation.	`var wert = 32 * 8;`
/	Mathematische Operation. Division.	`var wert = 32 / 8;`
<	Wertvergleich. Rückgabe True, wenn Wert kleiner ist oder False, wenn Wert größer ist.	`if(wert < 42) { }`
>	Wertvergleich. Rückgabe True, wenn Wert größer ist oder False, wenn Wert kleiner ist.	`if(wert > 42) { }`
<=	Wertvergleich. Rückgabe True, wenn Wert kleiner oder gleich ist oder False, wenn Wert größer ist.	`if(wert <= 42) { }`
>=	Wertvergleich. Rückgabe True, wenn Wert größer oder gleich ist oder False, wenn Wert kleiner ist.	`if(wert >= 42) { }`
+	Verkettung von Zeichenfolgen. ASP.NET erkennt, ob es sich um eine Verkettung oder um die mathematische Operation Addition	`var zeichenfolge = "Hello" +  "World";`
++	Inkrement. Addiert 1 zum vorherigen Wert der Variablen.	`wert++;`
--	Dekrement. Subtrahiert 1 vom vorherigen Wert der Variablen.	`wert--;`
+=	Addiert angegebenen Wert zur Variablen.	`wert += 4;`
-=	Subtrahiert angegebenen Wert von Variablen.	`wert -= 4;`
.	Punkt. Teilt Objekte in deren Methoden und Eigenschaften.	`var ganzzahl = Convert.ToInt32("42");`
()	Runde Klammern fassen Ausdrücke zusammen.	`var ganzzahl = (32 + 8) * 2;`
[]	Eckige Klammern. Zugriff auf Inhalte von Arrays, Listen und Collections.	`var ganzzahl = arrayVonGanzzahlen[2];`
!	Nicht. Vertauscht die Wertigkeit eines boolschen Ausdrucks. Ist der boolschen Wert True, ist die Ausgabe False und umgekehrt.	`bool ausdruck = false;` `bool ausdruckNicht = !ausdruck;`
&&	Logische Und-Verknüpfung.	`if(wert < 42 && wert > 8) { }`
||	Logische Oder.Verknüpfung.	`if(wert < 8 || wert == 12) { }`

Abbildung 7.9.1 Unterstützte Operator von Razor

[14] Vgl. (Microsoft ASP.NET, 2014), Operators

7.10. if-Anweisungen

Razor unterstützt serverseitige `if`-Anweisungen. `If`-Anweisungen können als einfache Anweisungen auftreten. Dabei wird serverseitig abgefragt, ob die Bedingung erfüllt ist. Ist dies der Fall, wird der innerhalbliegende Quellcode ausgeführt.

```
@{
    // Einfache if-Anweisung
    bool zeigeDatum = true;

    if (zeigeDatum)
    {
        <p>@DateTime.Now</p>
    }
}
```

Quellcode 7.10.1 Einfache `if`-Anweisung

`If`-Anweisungen können durch eine `else`-Anweisung ergänzt werden. Der innerhalb der `else`-Anweisung befindlichen Quellcode wird ausgeführt, wenn die Anweisung nicht erfüllt sein sollte.

```
@{
    // Einfache if-else-Anweisung
    bool zeigeDatum = true;

    if (zeigeDatum)
    {
        <p>@DateTime.Now</p>
    }
    else
    {
        <p>Datum wird nicht angezeigt.</p>
    }
}
```

Quellcode 7.10.2 Einfache `if-else`-Anweisung

`If`-Anweisungen lassen sich, um mehrere Bedingungen abzubilden, ineinander schachteln. Dabei wird in die `else`-Anweisung eine erneute `if-else`-Anweisung eingefügt.[15]

[15] Vgl. (Microsoft ASP.NET, 2014), Conditional Logic and Loops

```
@{
    // Geschachtelte if-else-Anweisung
    bool zeigeDatum = true;
    bool zeigeUhrzeit = true;

    if (zeigeDatum)
    {
        <p>@DateTime.Now</p>
    }
    else if (zeigeUhrzeit)
    {
        <p>@DateTime.Now.ToShortTimeString()</p>
    }
    else
    {
        <p>Datum und Uhrzeit werden nicht angezeigt.</p>
    }

}
```

Quellcode 7.10.3 Geschachtelte `if-else`-Anweisung

7.11. switch-Anweisung

Um noch tiefergehende Bedingungen abzubilden, sollte auf die `switch`-Anweisung zurückgegriffen werden. Diese wird von Razor serverseitig unterstützt und kann in nur einem Code-Blocks unendlich viele Bedingungen aufnehmen.

```
@{
    // switch-Anweisung
    int ganzzahl = 2;

    switch (ganzzahl)
    {
        case 1:
            <p>Die Zahl lautet: @ganzzahl</p>
            break;
        case 2:
            <p>Die Zahl lautet: @ganzzahl</p>
            break;
        case 3:
            <p>Die Zahl lautet: @ganzzahl</p>
            break;
        case 4:
            <p>Die Zahl lautet: @ganzzahl</p>
            break;
        default:
            <p>Die Zahl hat keinen gültigen Wert</p>
            break;
    }
}
```

Quellcode 7.11.1 Beispiel einer `switch`-Anweisung

An die `switch`-Anweisung wird der Parameter übergeben. Ist der Wert des Parameters gleich dem Wert eines der Fälle (`case`), wird dieser Code durchlaufen. Sollte der Parameter niemals dem Wert eines Falles (`case`) entsprechen, dann wird die Standard-Anweisung (`default`) ausgeführt. [16]

7.12. for-Schleife

In der Programmierung kommt es nicht selten vor, dass Anweisungen bis zu einem gewissen Zustand ausgeführt werden müssen. Dies wird über `for`-Schleifen abgebildet, die von Razor unterstützt werden.[17]

```
@{
    // for-Schleife

    for (int i = 0; i < 20; i++)
    {
        <p>Zahl ist: @i</p>
    }
}
```

Quellcode 7.12.1 Beispiel einer einfachen `for`-Schleife

7.13. foreach-Schleife

Speziell für Collections, nutzt Razor die `foreach`-Schleife. Diese Schleife durchläuft jedes Objekt einer Collection. `Foreach`-Schleifen haben den Vorteil, dass kein Zähler zu inkrementieren ist. Collections können als Array, Listen oder Dictionaries auftreten (siehe Kapitel 7.15 Collections, Seite 26).[18]

```
@{
    // foreach-Schleife
    string[] sammlungVonVornamen = { "Peter", "Max", "Florian" };

    foreach (string vorname in sammlungVonVornamen)
    {
        <p>Der Vorname lautet: @vorname</p>
    }
}
```

Quellcode 7.13.1 `foreach`-Schleife mit Zeichenfolgen-Array (`string[]`)

[16] Vgl. (Microsoft ASP.NET, 2014), Conditional Logic and Loops
[17] Vgl. (Microsoft ASP.NET, 2014), Looping Code
[18] Vgl. (Microsoft ASP.NET, 2014), Looping Code

7.14. while-Schleife

Eine weitere Integration von Schleifen, bildet die `while`-Schleife ab. Diese Schleife bekommt als Parameter eine Bedingung und durchläuft den Programmcode so lange, bis diese Anweisung nicht mehr erfüllt ist.[19]

```
@{
    // while-Schleife
    int ganzzahl = 20;

    while (ganzzahl > 5)
    {
        ganzzahl--;
    }
}
```

Quellcode 7.14.1 Beispiel einer `while`-Schleife

7.15. Collections

Eine Collection ist eine Sammlung von Objekten des gleichen Typs. Diese kommen meist aus dem Model der ASP.NET-Webanwendung. Eine Tabelle in einer Datenbank, die alle Benutzer eines Systems enthält, ergibt eine Collection vom Typ Benutzer, die beispielsweise drei Benutzer beinhaltet. Es gibt drei Arten von Collections die bei Razor eingesetzt werden können.[20]

7.15.1. Array

Ein Array speichert eine fest vorgegebene Anzahl an Objekten eines Datentyps. Über den Index des Arrays wird auf die einzelnen Einträge zugegriffen. Das erste Objekt hat den Index 0.

```
@{
    // Array als Collection mit der Größe 3
    string[] arrayMitVornamen = { "Peter", "Max", "Florian" };

    // Im Browser wird Max (Index 1) ausgegeben
    <p>arrayMitVornamen[1]</p>
}
```

Quellcode 7.15.1 Array des Datentyps Zeichenfolge (`string`)

[19] Vgl. (Microsoft ASP.NET, 2014), Looping Code
[20] Vgl. (Microsoft ASP.NET, 2014), Collection Objects (Arrays and Dictionaries)

7.15.2. List

Eine Liste hat bei der Instanziierung keine vorgegebene Größe. Es können Objekte hinzugefügt und entfernt werden, wodurch sich die Größe der Liste dynamisch anpasst. Über den Index wird ebenfalls auf die einzelnen Objekte der Liste zugegriffen.

```
@{
    // List als Collection
    List<string> arrayMitVornamen = new List<string>();
    arrayMitVornamen.Add("Peter");
    arrayMitVornamen.Add("Max");
    arrayMitVornamen.Add("Florian");

    // Im Browser wird Florian (Index 2) ausgegeben
    <p>arrayMitVornamen[2]</p>
}
```

Quellcode 7.15.2 List des Datentyps Zeichenfolge (`string`)

7.15.3. Dictionary

Ein Verzeichnis hat wie die Liste keine Größe bei der Instanziierung. Es besteht immer aus einem sogenannten Key-Value-Pair-Objekte, welches selbst zwei Datentypen enthält, das Key- und das Value-Attribut. Diese Datentypen können beim Instanziieren selbst festgelegt werden. Ein Verzeichnis hat den Vorteil, dass es nach dem Key- und dem Value-Attribut durchsucht werden kann. Dies ist bei einem Array und einer Liste nicht möglich.

```
@{
    // Dictionary als Collection
    Dictionary<string, string> arrayMitBenutzern = new Dictionary<string, string>();
    arrayMitBenutzern.Add("Peter", "Müller");
    arrayMitBenutzern.Add("Max", "Mustermann");
    arrayMitBenutzern.Add("Florian", "Meier");

    // Im Browser wird Peter (Index mit Value-Attribut "Müller") ausgegeben
    <p>arrayMitBenutzern["Müller"]</p>
}
```

Quellcode 7.15.3 Dictionary mit Zeichenfolgen (`string, string`)

7.16. Fehlerbehandlung

Serverseitig können diverse Fehler innerhalb des Quellcodes auftreten. Beispielsweise wird versucht über den Controller ein Datensatz im Model zu löschen. Ist der Server jedoch nicht berechtigt, Datensätze zu löschen, wird eine Ausnahme (`Exception`) geworfen (`throw`). Wird diese Ausnahme nicht abgefangen, dann wird der Fehler an den Browser gesendet und dieser stellt den Fehler dar.

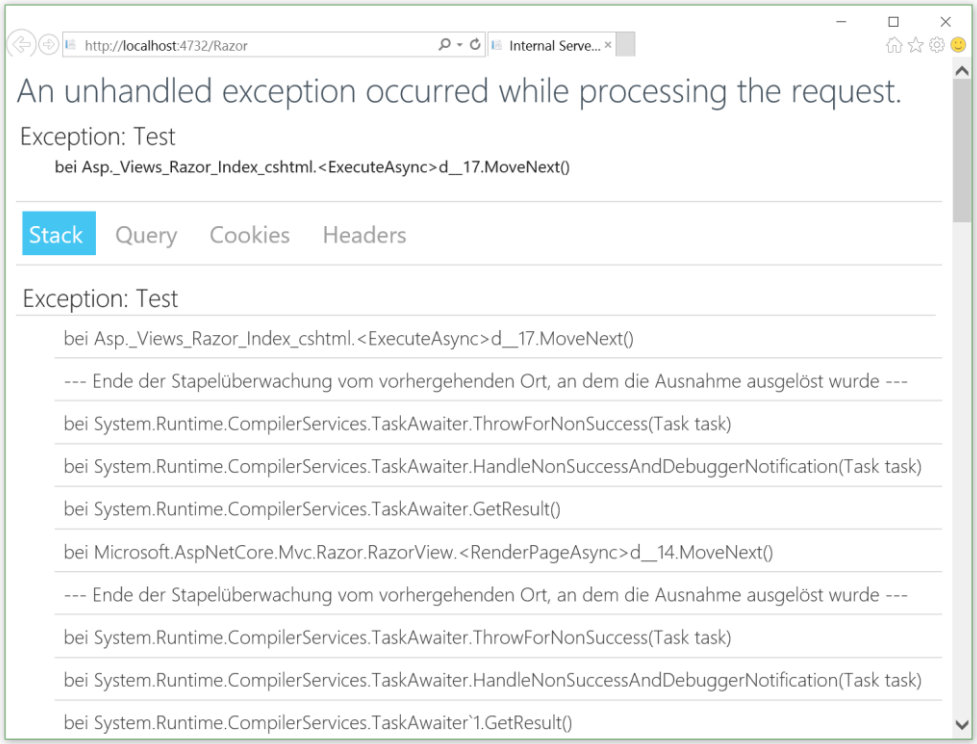


Abbildung 7.16.1 Unbehandelte, serverseitige `Exception`

Um diese Fehler korrekt abzufangen, ist ein sogenanntes `Exception`-Handling erforderlich. Den serverseitigen Programmcode, der erfolgreich ausgeführt werden soll, umschließt man mit einem `try-catch`-Anweisung. Tritt innerhalb des `try`-Blocks eine Ausnahme (`Exception`) auf, wird diese über den `catch`-Block abgefangen. Dadurch wird verhindert, dass die Applikation abstürzt.[21]

```
@{
    try
    {
        <p>Dieser Code wird versucht erfolgreich gerendert zu werden.</p>
    }
    catch (Exception error)
    {
        <p>@error.Message</p>
    }
}
```

Quellcode 7.16.1 `try-catch`-Anweisung für `Exception`-Handling

[21] Vgl. (Microsoft ASP.NET, 2014), Handling Errors

7.17. Zugriff auf Controller

Bei Razor und ASP.NET 5 MVC 6 RC1 gibt es drei Möglichkeiten über die View auf die Daten des Controllers zuzugreifen.[22]

7.17.1. ViewBag

`ViewBag` ist ein dynamisches Objekt, welchem Eigenschaften per Kaskadierung zugeordnet werden.

```
public IActionResult Index()
{
    ViewBag.Greeting = "Hello World!";
    return View();
}
```

Quellcode 7.17.1 `ViewBag`-Objekt mit Eigenschaft `Greeting`

Per Razor ist der Zugriff wie folgt möglich:

```
@{
    <p>@ViewBag.Greeting</p>
}
```

Quellcode 7.17.2 Zugriff auf Eigenschaft `Greeting` per Razor

Im Element `<p>` stünde bei Ausführung die Zeichenfolge `"Hello World!"`.

7.17.2. ViewData

`ViewData` ist im Gegensatz zu `ViewBag` ein Verzeichnis von Zeichenfolgen (`string`), die Werte enthalten können. Nachteilig ist, dass eine Typkonvertierung notwendig ist, wenn beispielsweise Ganzzahlen (`int`) übergeben werden. `ViewData` wurde deshalb größtenteils von `ViewBag` abgelöst.

```
public IActionResult Index()
{
    ViewData["Greeting"] = "Hello World!";
    return View();
}
```

Quellcode 7.17.3 `ViewData`-Verzeichnis mit Zeichenfolge `Greeting`

[22] Vgl. (Habib, 2012), What is ViewData, ViewBag and TempData?

Der Zugriff per Razor funktioniert wie folgt:

```
@{
    <p>@ViewData["Greeting"]</p>
}
```

Quellcode 7.17.4 Zugriff auf Zeichenfolge `Greeting` per Razor

Im Element `<p>` stünde bei Ausführung ebenfalls die Zeichenfolge `"Hello World!"`.

7.17.3. TempData

Der Unterscheid zu `ViewBag` und `ViewData` liegt darin, dass `TempData`-Einträge keine `NULL`-Werte erhalten, wenn eine Umleitung durchgeführt wurde.

```
public IActionResult Index()
{
    TempData["Greeting"] = "Hello World!";
    return View();
}
```

Quellcode 7.17.5 `TempData`-Verzeichnis mit Zeichenfolge `Greeting`

Der Zugriff per Razor ist analog zu `ViewData`.

```
@{
    <p>@TempData["Greeting"]</p>
}
```

Quellcode 7.17.6 Zugriff auf Zeichenfolge `Greeting` per Razor

Auch hier stünde im Element `<p>` bei Ausführung die Zeichenfolge `"Hello World!"`.

8. Web-Apps entwickeln mit Visual Studio

Diese Kapitel behandelt die eigentliche Entwicklung einer Web-App auf Basis von ASP.NET 5 MVC 6 RC 1 mithilfe von Visual Studio Enterprise 2015. Das Vorgehen wird anhand einer Beispiel-Applikation beschrieben, die zu Beginn inhaltslos ist. Zum Ende des Kapitels beinhaltet die Web-App Models, Controller und Views und ist auf der Microsoft Azure Cloud Plattform[23] veröffentlicht.

8.1. Erstellen eines ASP.NET 5 MVC 6 RC1 Projektes

Nach dem Start von Visual Studio Enterprise 2015 erscheint der Willkommensbildschirm. Über den Menüpunkt `Datei` > `Neu` > `Projekt` öffnet sich ein Fenster, in dem die gewünschte Art des Projektes und das zu verwendende Framework ausgewählt werden.

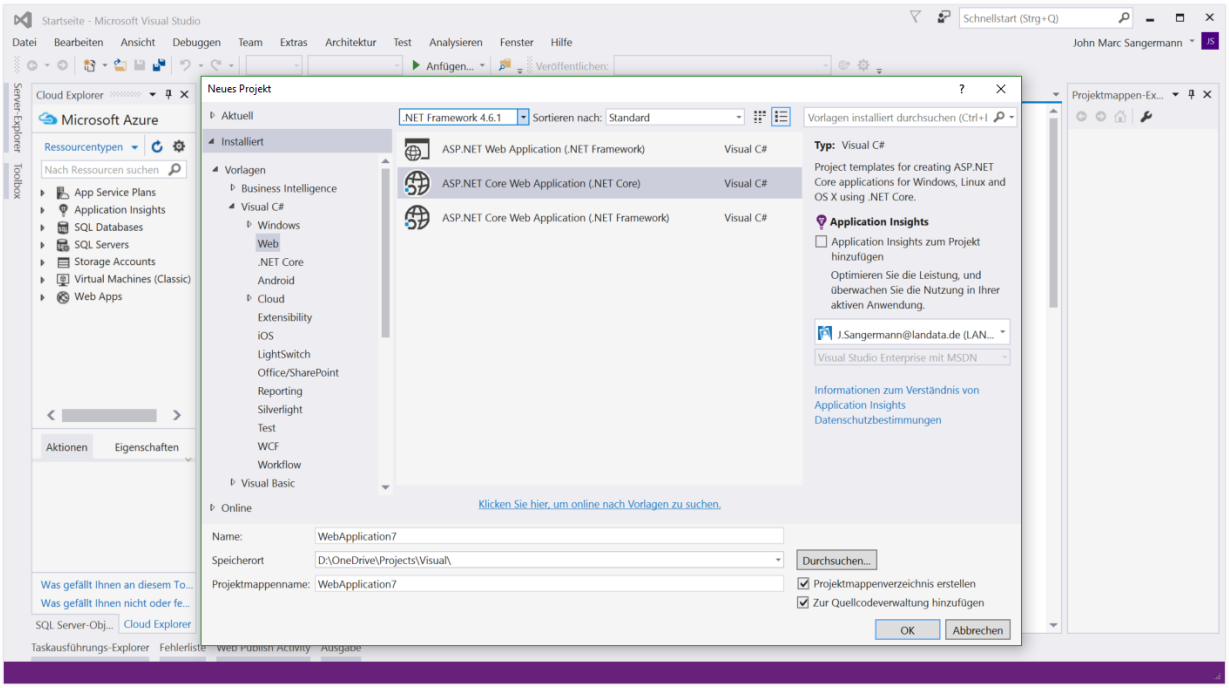


Abbildung 8.1.1 Neues Projekt in Visual Studio Enterprise 2015 anlegen

Mit `OK` wird das Erstellen der Projektmappe fortgeführt. Im darauffolgenden Fenster wird eine Projektvorlage ausgewählt. Bei Auswahl von `Leer` wird ein Blanko-Gerüst einer ASP.NET 5 MVC 6 RC1 Web-App erzeugt. Wird `Web Application` ausgewählt, kann direkt eine Benutzer-Authentifizierungsmethode ausgewählt werden (siehe Kapitel 8.11 Authentifizierung, Seite 63). Web-API wird in dieser Arbeit nicht behandelt.[24]

[23] Siehe (Microsoft Azure, 2016), Ihre App. Ihr Framework. Ihre Plattform. Platz für alles.
[24] Vgl. (Microsoft ASP.NET, 2016), Getting started with ASP.NET Core MVC and Visual Studio

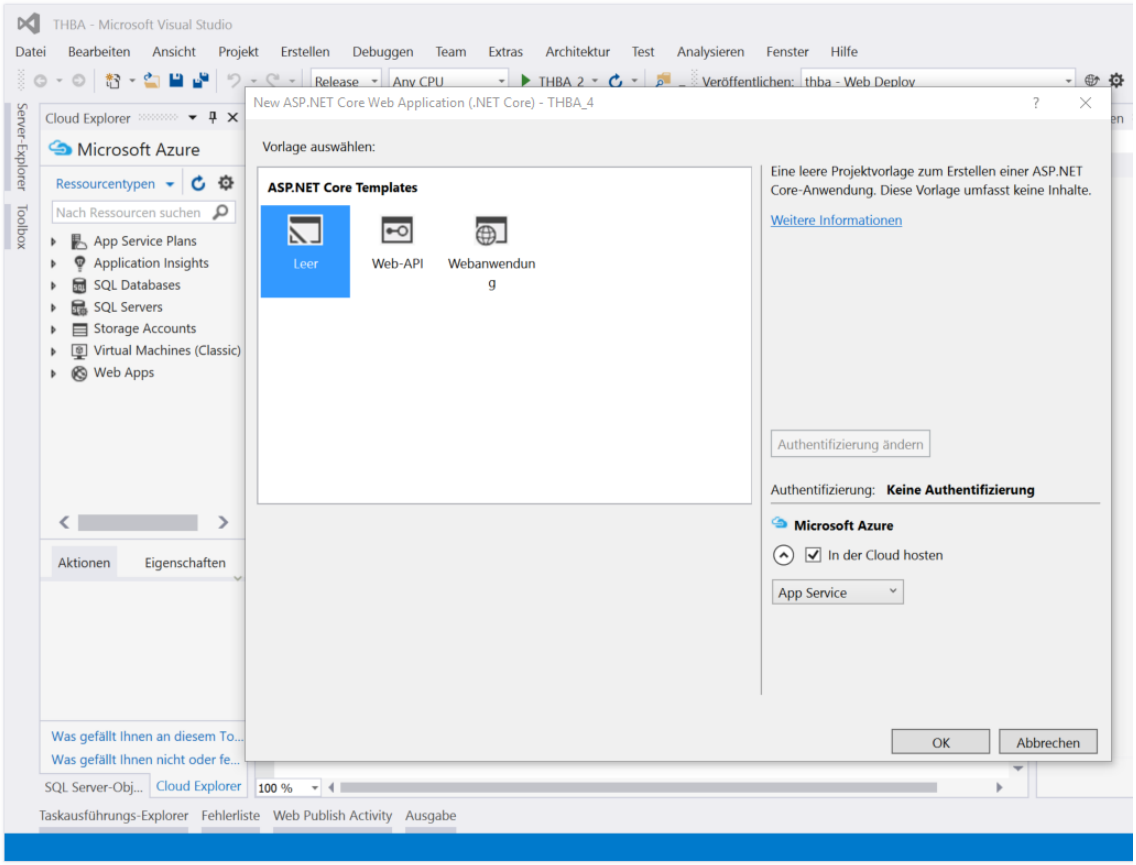


Abbildung 8.1.2 Auswahl der Projektlage

8.2. Pakete eines ASP.NET 5 MVC 6 RC1 Projektes

ASP.NET 5 MVC 6 RC1 Projekte bestehen aus Paketen. Es gibt zwei Arten von Paketen. Zum einen NuGet und zum anderen Bower. Die beiden Paket-Manager installieren und verwalten die gewünschten Pakete und Erweiterungen des ASP.NET 5 MVC 6 RC1 Projektes und installieren abhängige Bibliotheken automatisch mit. Als Beispiele seien hier die Javascript Bibliothek JQuery[25] und das Framework Bootstrap[26] genannt, welche in der erstellten Web-App zum Einsatz kommen werden. Bootstrap ist in diesem Fall abhängig von JQuery. Würde man Bootstrap installieren, dann würde JQuery ebenfalls mit installiert.

8.2.1.NuGet

Bei NuGet handelt es sich um einen Paketmanager für das Microsoft .NET-Framework. Dieser Manager bietet die Möglichkeit, Pakete in einem Projekt zu installieren und zu verwalten. Die kompatiblen Pakete werden in dem Paket-Repository NuGet Gallery von Autoren veröffentlicht.[27] Auf diesen Repository kann direkt aus Visual Stu-

[25] Siehe (jQuery, 2016)
[26] Siehe (Bootstrap, 2016)
[27] Vgl. (NuGet, 2016), What is NuGet?

dio Enterprise 2015 zugegriffen werden. Über einen Rechtsklick auf das Projekt können die NuGet-Pakete verwaltet werden. Öffnet man den NuGet-Paket-Manager ist zu erkennen, dass die notwendigen Pakete für eine ASP.NET 5 MVC 6 RC1 Web-App bereits installiert sind. Über den Menüeintrag `Durchsuchen` lassen sich weitere Pakete suchen und installieren.

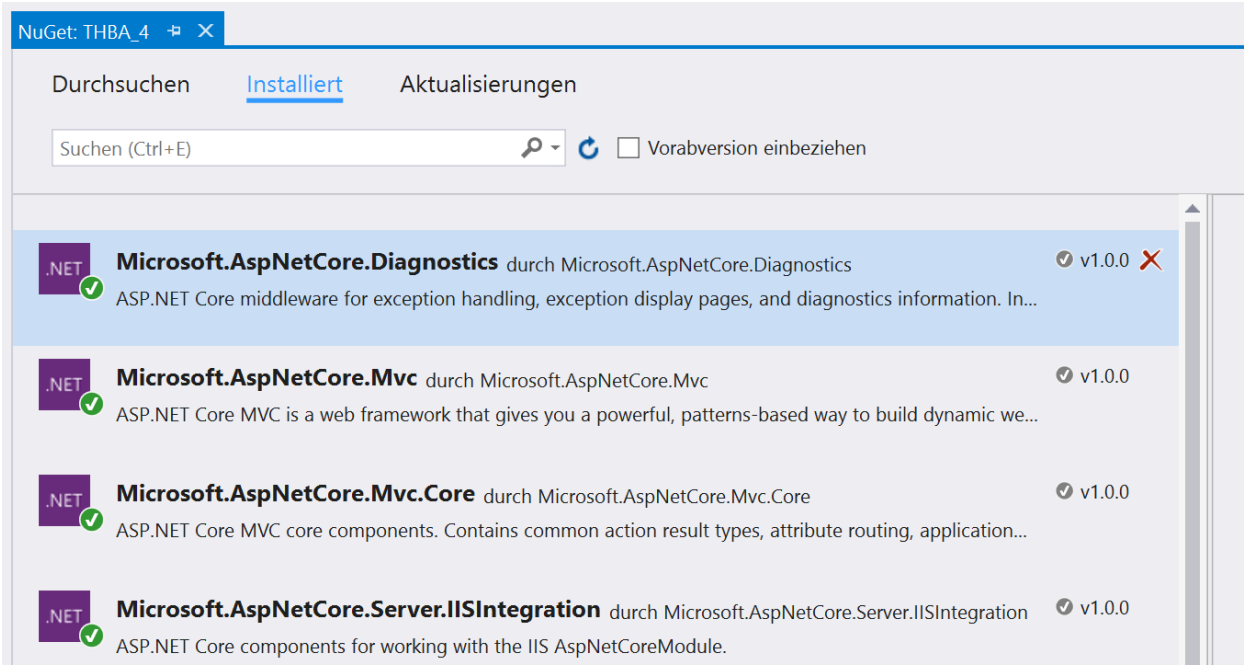


Abbildung 8.2.1 NuGet-Paket-Manager in Visual Studio Enterprise 2015

Der NuGet-Pakete-Manager besitzt zusätzlich eine Konsole über die die Pakete ebenfalls verwaltet werden können. Die Konsole wird über den Menüeintrag `Extras` > `Nuget-Pakete-Manager` > `Pakete-Manager-Konsole`[28] aufgerufen.

8.2.2.Bower

Bower ist ebenfalls ein Paket-Manager, welcher jedoch nicht explizit für das Microsoft .NET-Framework entwickelt worden ist. Über Bower können einem ASP.NET 5 MVC 6 RC1 Projekt clientseitige Erweiterungen installiert und verwaltet werden.[29] Die Verwaltung der Pakete verläuft ähnlich wie bei NuGet-Paketen. Über einen Rechtsklick auf das Projekt, können auch hier die Bower-Pakete verwaltet werden. Über den Menüeintrag `Durchsuchen` lassen sich weitere Pakete installieren. Bower kann die Pakete-Manager-Konsole nutzen.

[28] Vgl. (NuGet, 2008), Managing Packages Using the Package Manager Console
[29] Siehe (Bower, 2016)

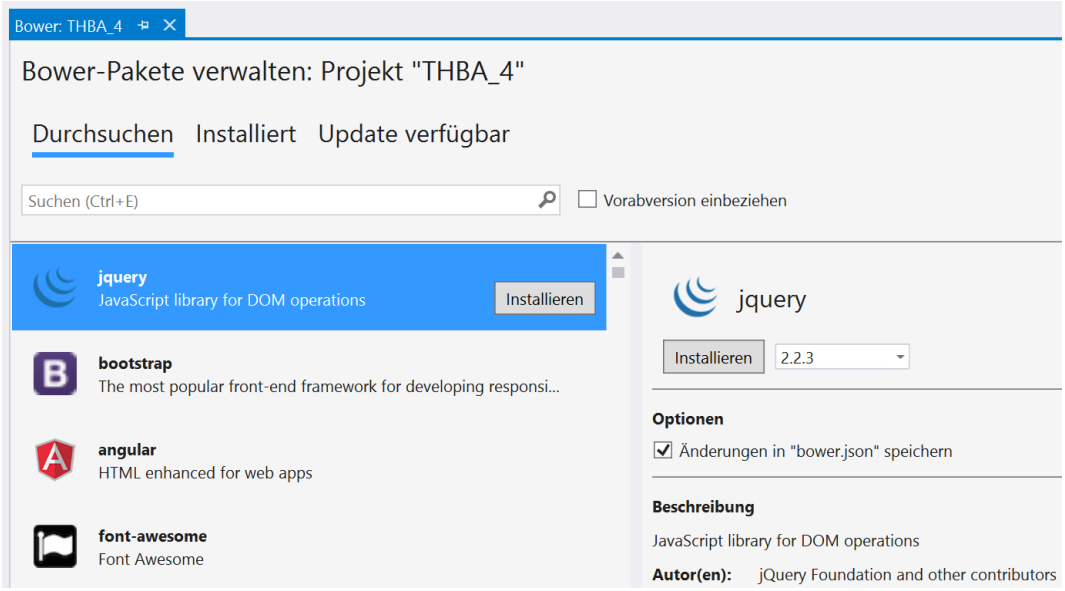


Abbildung 8.2.2 Bower-Paket-Manager in Visual Studio Enterprise 2015

8.3. Aufbau einer ASP.NET 5 MVC 6 RC1 Web-App

Nachdem das Grundgerüst erzeugt wurde, zeigt der Projektmappen-Explorer den Aufbau einer bereits lauffähigen ASP.NET 5 MVC 6 RC1 Webanwendung.

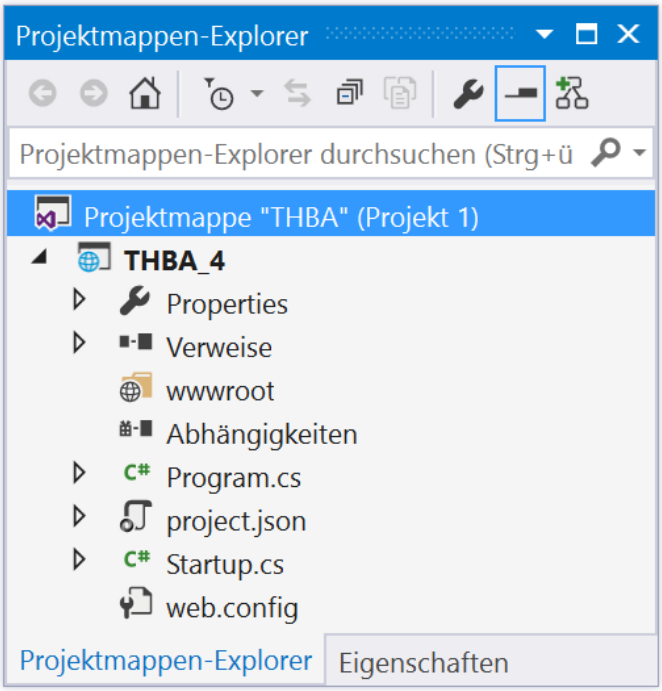


Abbildung 8.3.1 Aufbau eines Projektes im Projektmappen-Explorer

Über die Einstellungen (`Properties`) können Build- und Debugoptionen festgelegt werden, wie, ob eine Dokumentation als XML-Dokument erstellt wird oder der Browser

nach dem Hosten der Webanwendung automatisch startet etc. Die Profile zum Veröffentlichen der Web-App werden ebenfalls dort abgelegt (siehe Kapitel 8.14 Veröffentlichung, Seite 73). Die Verweise beinhalten die notwendigen Bibliotheken der Webanwendung. Beispielsweise sei hier die Bibliothek `Microsoft.AspNetCore.Server.Kestrel` genannt, die für das Ausführen der Webanwendung auf einem lokalen Webserver notwendig ist. Die Bibliotheken werden über NuGet- oder Bower-Pakete dem Projekt hinzugefügt (siehe Kapitel 8.2 Pakete eines ASP.NET 5 MVC 6 RC1 Projektes, Seite 32). Das Verzeichnis `wwwroot` beinhaltet alle zusätzlichen Dateien der Webanwendung. Hier zu finden sind die Skripte, Stylesheets, Bilder etc. Unter den Abhängigkeiten werden die Abhängigkeiten von Paketen gelistet. In der Datei `project.json`[30] werden alle Informationen und Abhängigkeiten zusammengefasst und übersichtlich dargestellt. Es können Informationen, wie Name, Version, Beschreibung, Copyright, Titel, Sprache, Autor ergänzt werden.

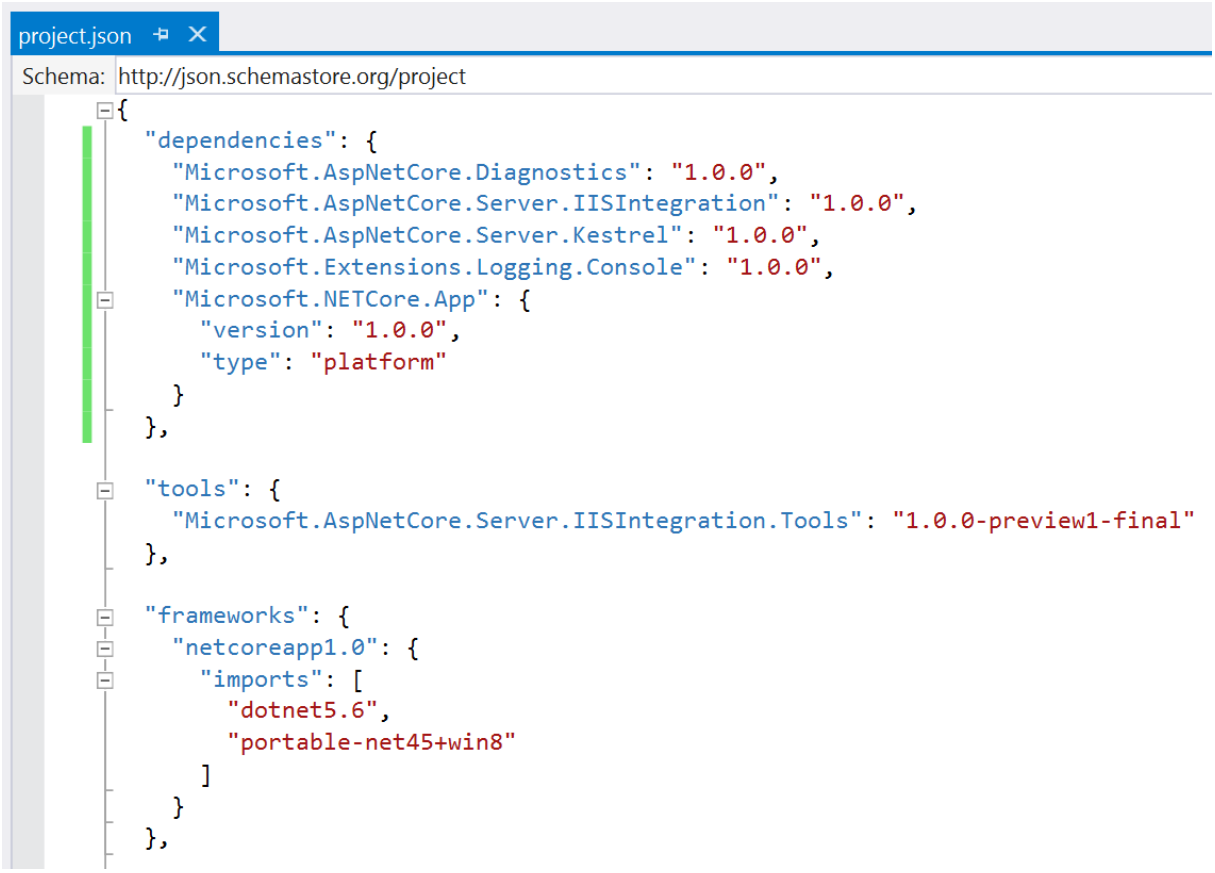


Abbildung 8.3.2 Snippet der `project.json`

Die `project.json` kann über die Informationen der `web.config` erweitert werden. In dieser lassen sich ergänzende Einstellungen der Anwendung hinterlegen. Die `web.config`[31] beinhaltet beispielsweise die Datenbankverbindungszeichenfolgen (`connectionstrings`).

[30] Vgl. (Microsoft Docs, 2016), project.json reference
[31] Vgl. (Microsoft Developer Network, kein Datum), Abschnitte der Konfigurationsdatei

Web.config

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  Weitere Informationen zum Konfigurieren der ASP.NET-Anwendung finden Sie unter
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <connectionStrings>
    <add name="DBConnectionString" connectionString="Data Source=SQLSERVER\MSSQLSERVER2014;
         Initial Catalog=DB;Persist Security Info=True;User ID=sa;Password=XXX"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.6.1" />
    <httpRuntime targetFramework="4.6.1" />
  </system.web>
</configuration>
```

Abbildung 8.3.3 Snippet der `web.config`

Mithilfe der Klasse `Program` und der darin enthaltenen Methode `Main` wird die Web-App gestartet.

```
using Microsoft.AspNetCore.Hosting;

namespace THBA_4
{
    0 Verweise
    public class Program
    {
        0 Verweise
        public static void Main(string[] args)
        {
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .Build();

            host.Run();
        }
    }
}
```

Quellcode 8.3.1 Program-Klasse einer ASP.NET 5 MVC 6 RC1 Anwendung

Die `Main`-Methode instanziiert einen Host über die `WebHostBuilder`-Klasse. Diese Instanz definiert über Methoden die Eigenschaften des Host. Die Methode `UseKestrel` definiert einen `Kestrel`-http-Server[32]. Dieser wird benötigt um die Webanwendung lokal ausführen zu können. Die Methode `UseContentRoot` definiert ein Verzeichnis, in

[32] Vgl. (Microsoft ASP.NET, 2016)

dem die Dateien der Webanwendung zu finden sind. Standardmäßig hat das Verzeichnis den Namen `wwwroot` (siehe Abbildung 8.3.1 Aufbau eines Projektes im Projektmappen-Explorer, Seite 34). Über die Methode `UseIISIntegration` ist es möglich, dass die Webanwendung auf dem Microsoft Internet-Informationen Services (IIS)[33] gehostet werden kann. Dies ist ebenfalls notwendig um die Web-App in der Microsoft Azure Cloud Plattform zu veröffentlichen. Die Methode `UseStartup<>` spezifiziert die Start-Klasse der Webapplikation. Die `Startup`-Klasse ist für jede Anwendung notwendig und implementiert zwei eigene Methoden. Die Methoden `ConfigureServices` und `Configure`.

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    0 Verweise
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 Verweise
    public void Configure(IApplicationBuilder app, IHostingEnvironment env,
        ILoggerFactory loggerFactory)
    {
        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
}
```

Quellcode 8.3.2 Klasse `Startup` mit benötigten Methoden

Auf die Methode `ConfigureServices` wird in den folgenden Kapiteln weiter eingegangen. Die Methode `Configure` implementiert die http-Endpunkte der Anwendung und wie die Anwendung auf die http-Anforderungen reagiert. Die folgende Abbildung zeigt den Quellcode an, der die Zeichenfolge `Hello World!` an den Browser zurückgibt. Wird die Anwendung über Visual Studio Enterprise 2015 gestartet, startet der `Kestrel`-Webserver und hostet die Anwendung. Die Ausgabe geschieht in einem Webbrowser.

[33] Siehe (Microsoft IIS, kein Datum), IIS Overview

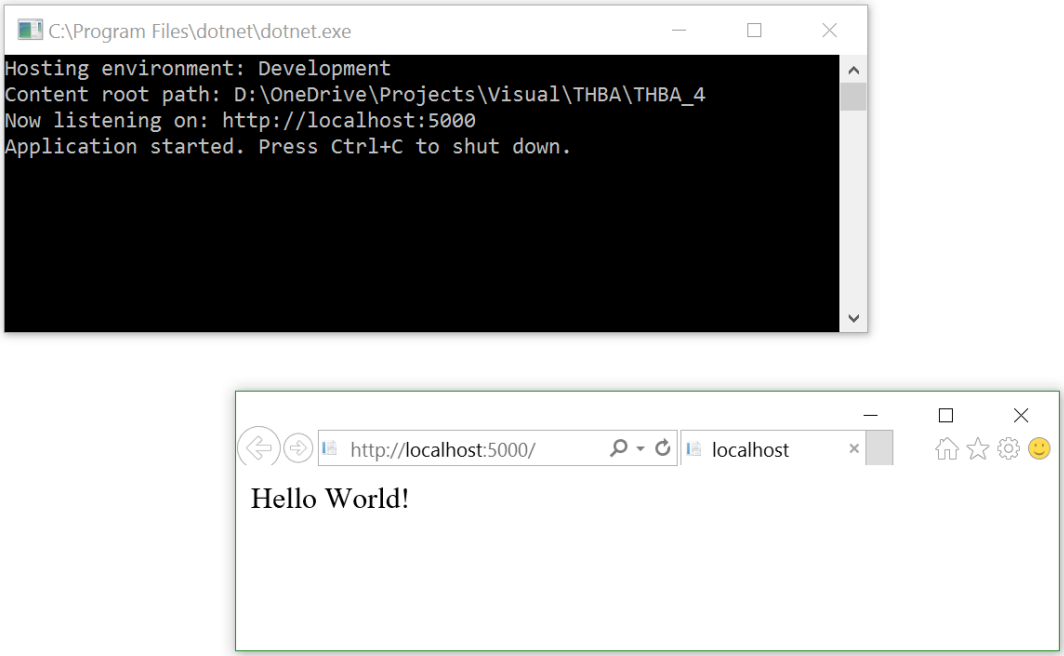


Abbildung 8.3.4 `Kestrel`-Webserver- und Browserausgabe der Anwendung.

Dies ist ein Beispiel für eine minimalistische ASP.NET 5 MVC 6 RC1 Web-App. Der Aufbau eines Projektes kann um weitere Dateien erweitert werden. Hierauf wird in folgenden Kapiteln weiter eingegangen.[34]

[34] Vgl. (Microsoft ASP.NET, 2016), Application anatomy

8.4. Logging

ASP.NET 5 MVC 6 RC1 hat eine eigebaute Unterstützung für das Logging von Ereignissen. Es gibt sechs unterschiedliche Stufen der Logging-Einträge:

5. Informationen (Präfix info)
6. Kritisch (Präfix crit)
7. Debuggen (Präfix dbug)
8. Fehler (Präfix fail)
9. Verfolgung (Präfix trce)
10. Warnung (Präfix warn)

Um das Logging in einer Webanwendung zu implementieren, wird das Interface `ILoggerFactory` verwendet, welches als Parameter an die `Configure`-Methode der `Startup`-Klasse übergeben wird (siehe Quellcode 8.3.2 Klasse `Startup` mit benötigten Methoden). Um das Logging auf der Konsole des Webservers zu verwenden, wird die Methode `AddConsole` Instanz des Interfaces `ILoggerFactory` ausgeführt. Hier ist zu beachten, dass das Log-Level standardmäßig nur Einträge oberhalb der Stufe `Informationen` ausgeben. Dies bedeutet, dass `Debuggen` und Verfolgung nicht ausgegeben werden. Übergibt man der Methode `AddConsole` den Parameter `LogLevel.Trace` dann werden alle Einträge in der Konsole ausgegeben.

Möchte man eigene Log-Einträge erstellen, muss eine Instanz des Interfaces `ILogger` instanziiert werden, welche auf die `Program`-Klasse verweist. Log-Einträge der jeweiligen Stufe lassen sich mit der zugehörigen Methode abbilden.[35] [36]

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(LogLevel.Trace);

    ILogger logger = loggerFactory.CreateLogger<Program>();

    logger.LogInformation("Info");
    logger.LogCritical("Kritisch");
    logger.LogDebug("Debuggen");
    logger.LogError("Fehler");
    logger.LogTrace("Verfolgung");
    logger.LogWarning("Warnung");

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

Quellcode 8.4.1 Anpassung an der `Configure`-Methode der `Startup`-Klasse

[35] Vgl. (Microsoft Developer Network, 2016), .NET-Grundlagen: Protokollieren mit .NET Core
[36] Vgl. (Microsoft ASP.NET, 2016), Logging

Startet man die Anwendung, werden die Ereignisse in der Konsole des Webservers ausgegeben.

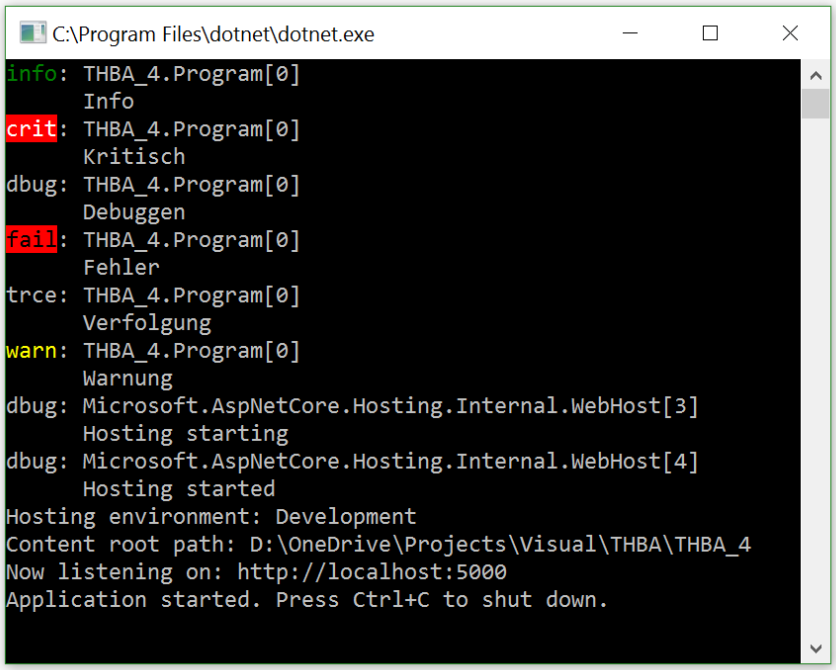


Abbildung 8.4.1 Konsolenausgabe mit Log-Einträgen

8.5. Tests

ASP.NET 5 MVC 6 RC1 Webanwendungen können über zwei Arten von Tests auf Funktion überprüft werden, Unit- und Integration-Tests. Das Framework xUnit.net kommt bei ASP.NET 5 MVC 6 RC1 Web-Apps zum Einsatz[37]. XUnit.net ist ein Open-Source-Testwerkzeug für das Microsoft .NET-Framework und ist über den NuGet-Pakete-Manager verwendbar.

8.5.1.Unit-Test mit xUnit.net

Unit-Test werden vom Entwickler selbst geschrieben und haben den Zweck einzelne Funktionen der Anwendung zu überprüfen. Sie enthalten nur einen kurzen Programmquellcode und sind schnell in der Ausführung. Unit-Tests liegen in der Verantwortung des Programmierers, welcher mithilfe von diesen Tests, als wichtig erachtete Programmteile testet. Beispielsweise, ob ein Rückgabewert einer Methode den Typ Ganzzahl (`int`) aufweist. Um einen Unit-Test einem ASP.NET 5 MVC 6 RC1 Projekt hinzuzufügen, macht man einen Rechtsklick auf die Projektmappen > `Hinzufügen` > `Neues Projekt` und wählt dort `Class Library (.NET Core)` aus.

[37] Siehe (xUnit.net, 2015)

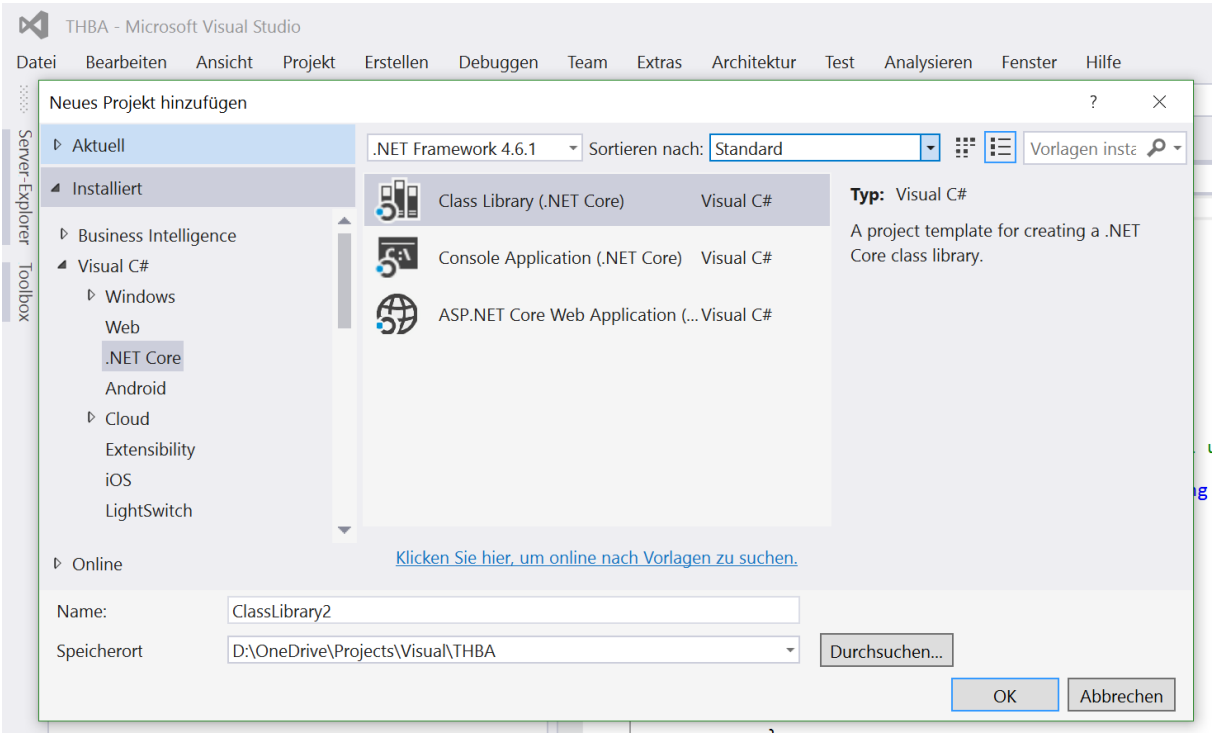


Abbildung 8.5.1 Projekt für Unit-Tests hinzufügen

Visual Studio Enterprise 2015 erstellt automatisch folgendes Projektgerüst.

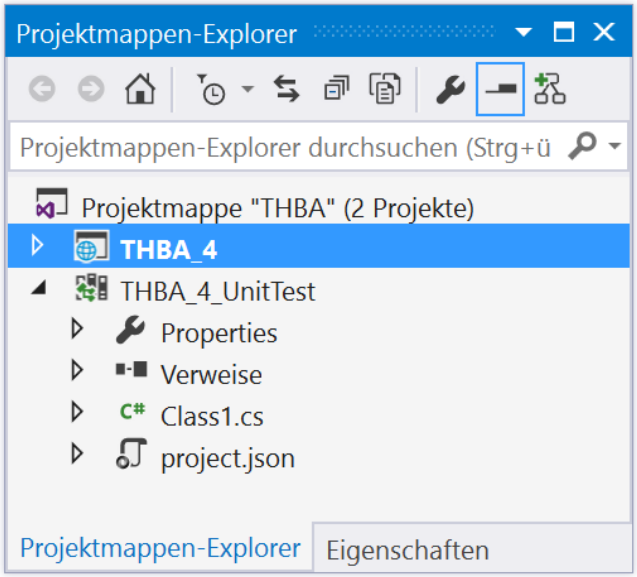


Abbildung 8.5.2 Grundgerüst Test-Projekt

XUnit.net wird dem Projekt über den NuGet-Pakete-Manager hinzugefügt. Da sich die Hauptanwendung und die Testanwendung in der gleichen Projektmappe befinden, ist es direkt möglich Unit-Tests zu erstellen, sonst wäre ein Verweis auf die Hauptanwendung, notwendig. Auf das Hinzufügen dieses Verweises wird hier nicht näher eingegangen. In der Hauptanwendung ist eine Klasse `UnitTest` ergänzt worden, die im

nächsten Schritt getestet wird. Die Methode der Klasse `istGanzzahlAlsZeichen-folge` gibt `True` zurück, wenn sich die Zeichenfolge (`string`) in eine Ganzzahl (`int`) umwandeln lässt. Andernfalls wird `False` zurückgegeben.

```
public class UnitTest
{
    // Methode versucht Zeichenfolge in Ganzzahl umzuwandeln
    2 Verweise | 0/2 wird übergeben
    public bool istGanzzahlAlsZeichenfolge(string wert)
    {
        try
        {
            Convert.ToInt32(wert);
            return true;
        }
        catch
        {
            return false;
        }
    }
}
```

Quellcode 8.5.1 Klasse für den Unit-Test

In dem Testprojekt wird die Klasse `Class1` um folgenden Quellcode ergänzt:

```
public class Class1
{
    private UnitTest utest = new UnitTest();

    // Dies sind einfache Unit-Tests mit xUnit.net-Framework
    [Fact]
    0 Verweise
    public void einfacherUnitTest1 ()
    {
        Assert.True(utest.istGanzzahlAlsZeichenfolge("42"));
    }

    [Fact]
    0 Verweise
    public void einfacherUnitTest2()
    {
        Assert.True(utest.istGanzzahlAlsZeichenfolge("Peter"));
    }
}
```

Quellcode 8.5.2 Klasse mit zwei Unit-Tests

Die Testanwendung prüft, ob sich die Zeichenfolgen (`string`) `"42"` und `"Peter"` in einer Ganzzahl (`int`) umwandeln lassen. Über den Menüpunkt `Test` > `Ausführen` > `Alle Tests` lassen sich die Unit-Tests ausführen. In diesem Fall ist Unit-Test 1 erfolgreich und Unit Test 2 fehlgeschlagen.

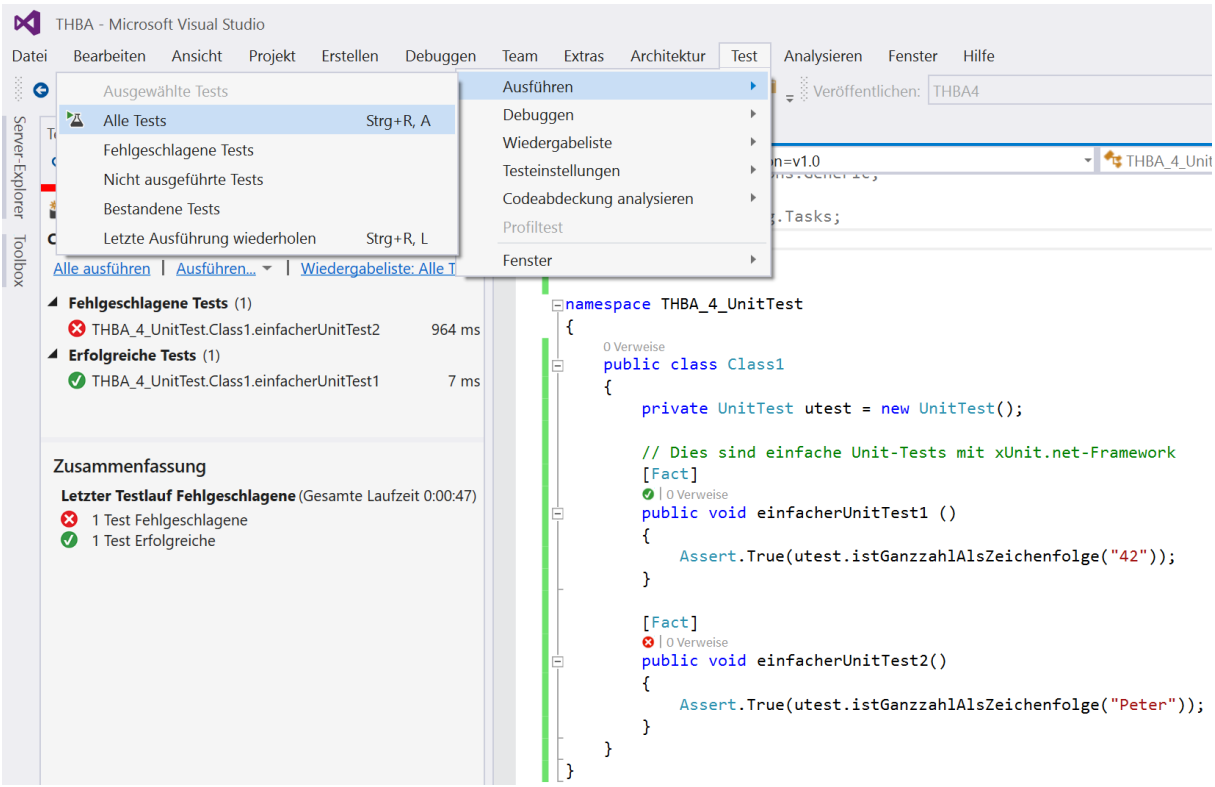


Abbildung 8.5.3 Erfolgreicher und fehlgeschlagener Unit-Test

8.5.2.Integration-Test mit xUnit.net

Bei Integration-Tests handelt es sich um Testmethoden, die das Zusammenspiel von Programmkomponenten testen. Integration-Tests sind meist ein Zusammenschluss von Unit-Test-Szenarien. Damit sind diese Tests aufwendig zu implementieren und besitzen eine längere Durchlaufzeit. Diese Tests demonstrieren, ob das System in einer Testumgebung produktiv eingesetzt werden kann. Beteiligte Komponenten können beispielsweise Datenbanken und Browserausgaben sein. Bei ASP.NET 5 MVC 6 RC1 Integration-Tests wird im Testprojekt ein eigener virtueller Testhost instanziiert. Integration-Test folgen dem Muster Arrange-Act-Assert. Im Bereich Arrange (Vorbereitung) werden der Testhost und der http-Client instanziiert. Im Abschnitt Act (Aktion) wird die zu testende Methode mit Parameter aufgerufen. Im Abschnitt Assert (Bestätigung) wird überprüft, ob die getestete Methode wie erwartet funktioniert. Ein Integration-Test lässt sich analog zu einem Projekt hinzufügen wie ein Unit-Test (siehe Kapitel 8.5.1 Unit-Test mit xUnit.net, Seite 40). [38]

[38] Vgl. (Microsoft Developer Network, 2016), Create unit test projects and test methods

In dem Testprojekt für den Integration-Test wird die Klasse `Class1` um den folgenden Quellcode ergänzt:

```
public class Class1
{
    private readonly TestServer _server;
    private readonly HttpClient _client;

    0 Verweise
    public Class1()
    {
        // Arrange; Testhost und HTTP-Client werden instanziiert
        _server = new TestServer(new WebHostBuilder()
            .UseStartup<Startup>());
        _client = _server.CreateClient();
    }

    [Fact]
    0 Verweise
    public async Task einfacherIntegrationTest1()
    {
        // Act; Index Ressource wird aufgerufen
        var response = await _client.GetAsync("/");
        response.EnsureSuccessStatusCode();
        var responseString = await response.Content.ReadAsStringAsync();

        // Assert; Test, ob "Response Hello World!" entspricht
        Assert.Equal("Hello World!", responseString);
    }

    [Fact]
    0 Verweise
    public async Task einfacherIntegrationTest2()
    {
        // Act; Index Ressource wird aufgerufen
        var response = await _client.GetAsync("/");
        response.EnsureSuccessStatusCode();
        var responseString = await response.Content.ReadAsStringAsync();

        // Assert; Test, ob "Response Hello to World!" entspricht
        Assert.Equal("Hello to World!", responseString);
    }
}
```

Quellcode 8.5.3 Klasse mit zwei Integration-Tests

`Alle Tests` lassen sich analog zu den Unit-Tests ausführen. In diesem Fall ist Integration-Test 1 erfolgreich und Integration-Test 2 schlägt fehl.

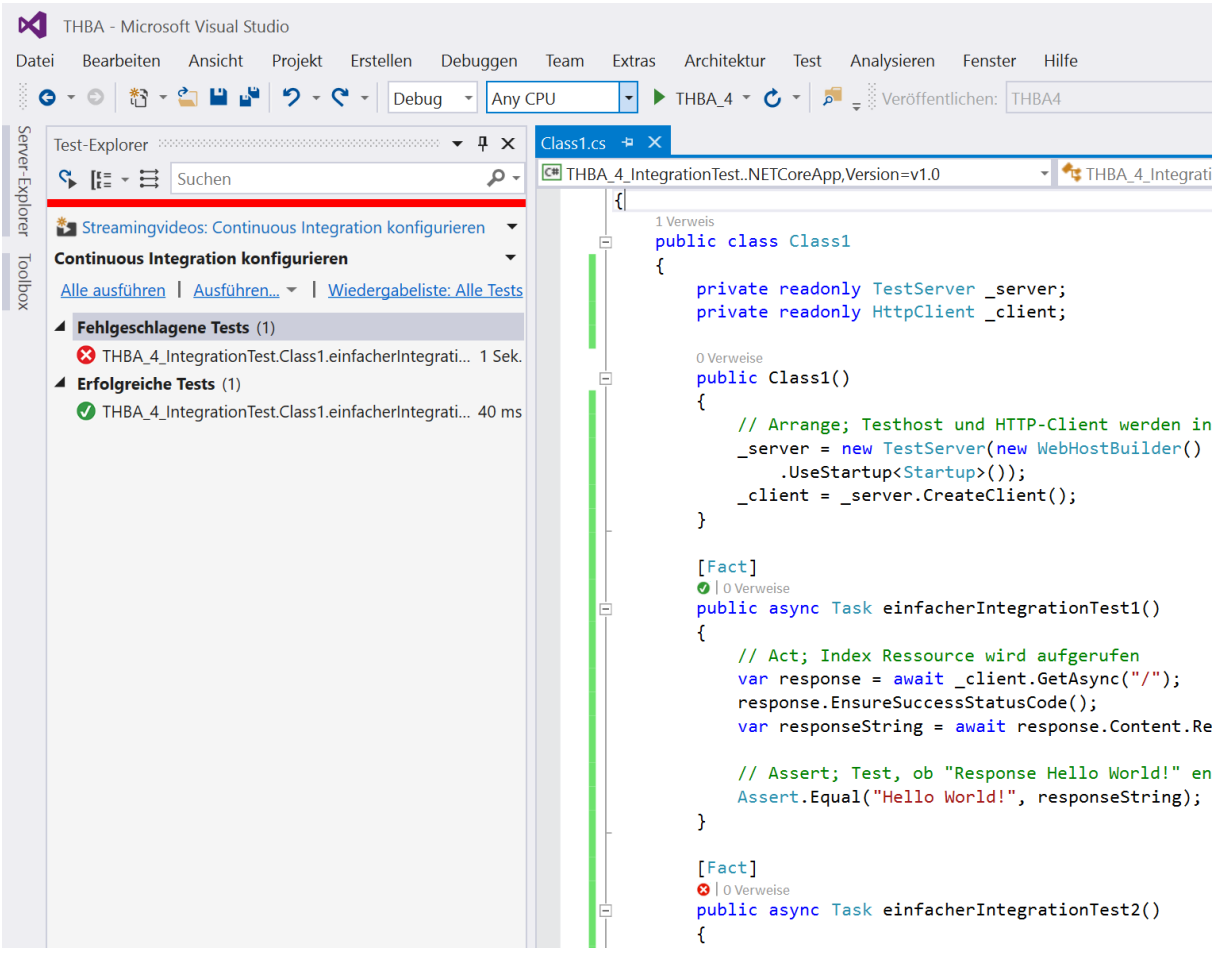


Abbildung 8.5.4 Erfolgreicher und fehlgeschlagener Integration-Test

8.6. MVC-Paradigma implementieren

Um das MVC-Paradigma in einer ASP.NET 5 MVC 6 RC1 Web-App zu implementieren, werden die Ordner Models, Views und Controllers in der Ordnerstruktur des Projektes ergänzt. Dies ist keine Pflicht, aber eine Empfehlung, um die Übersichtlichkeit des Projektes zu erhalten.

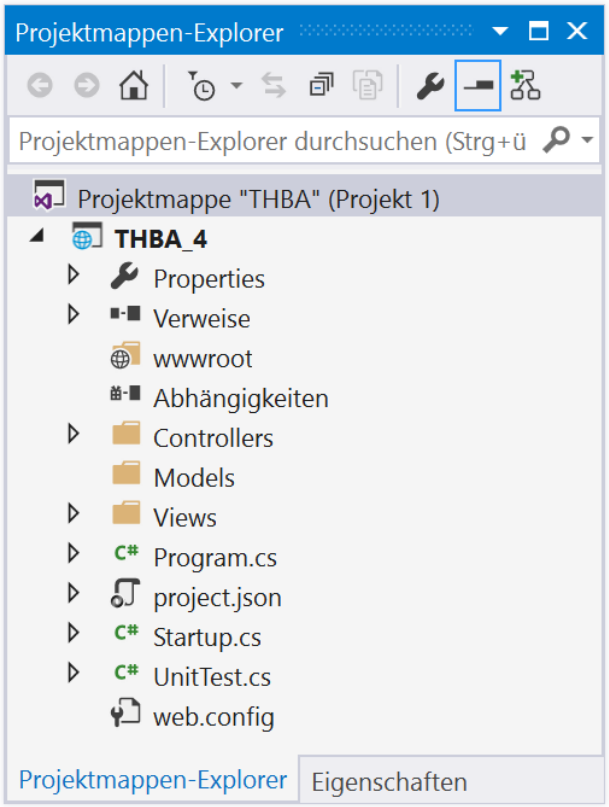


Abbildung 8.6.1 Ordnerstruktur mit MVC-Ordnern

Über den NuGet-Pakete-Manager wird die notwendige Bibliothek `Microsoft.AspNetCore.MVC` installiert (siehe 8.2.1 NuGet, Seite 32). Daraufhin werden an der Klasse `Startup` folgende Quellcodeänderungen durchgeführt:

```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    0 Verweise
    public void ConfigureServices(IServiceCollection services)
    {
        // Add framework services.
        services.AddMvc();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 Verweise
    public void Configure(IApplicationBuilder app, IHostingEnvironment env,
        ILoggerFactory loggerFactory)
    {
        app.UseMvc(routes =>
        {
            routes.MapRoute(
                name: "default",
                template: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}
```

Quellcode 8.6.1 Code der `Startup`-Klasse für MVC-Funktionalität

In der Methode `ConfigureServices` wird das Interface MVC vom Typ `IServiceCollection` über `services.AddMvc()` implementiert. Jegliche Services einer ASP.NET 5 MVC 6 RC1 Webanwendung werden über diese Methode integriert (siehe 8.3 Aufbau einer ASP.NET 5 MVC 6 RC1 Web-App, Seite 34). Die Änderungen an der Methode `Configure` erstellen das MVC-Routing. Hiermit ist gemeint, dass beim Aufruf der Seite standardmäßig der Controller `Home` und deren View `Index` aufgerufen wird. Optionale Parameter können über die `id` übergeben werden.

8.7. Controller hinzufügen

Über einen Rechtsklick auf den Ordner `Controller` > `Hinzufügen` > `Neues Element`, wird eine MVC-Controllerklasse der Anwendung hinzugefügt. Visual Studio Enterprise 2015 setzt automatisch den Namen `HomeController.cs` ein, da es sich hierbei um den Standard-Controller handelt.

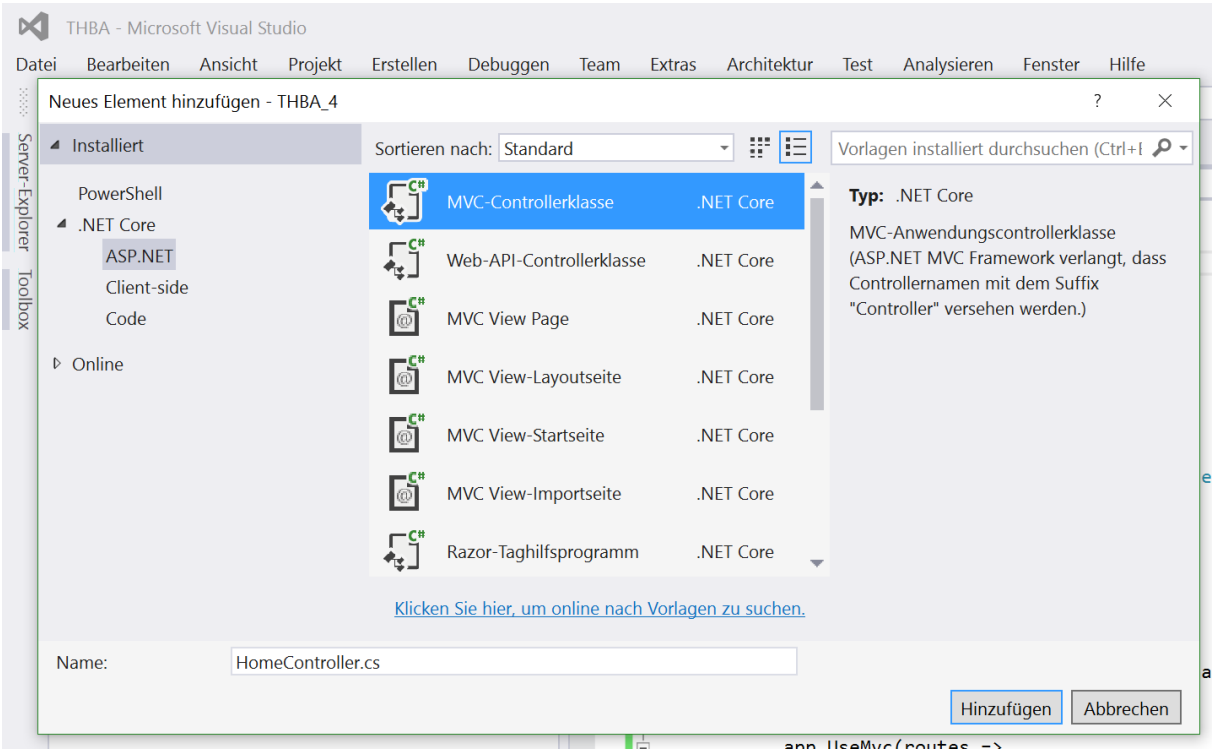


Abbildung 8.7.1 Hinzufügen einer MCV-Controllerklasse

Nach dem Erstellen des `HomeController`, hat dieser folgenden Inhalt:

```
public class HomeController : Controller
{
    // GET: /<controller>/
    0 Verweise
    public IActionResult Index()
    {
        return View();
    }
}
```

Quellcode 8.7.1 Code des Controllers `HomeController`

Wird der `HomeController` aufgerufen, wird die Methode `Index` getriggert, welche in diesem Fall als Instanz des Interfaces `IActionResult` eine View mit gleichem Namen zurückgibt. Da die View `Index` noch nicht vorhanden ist, muss diese anschließend erstellt werden.[39]

8.8. View hinzufügen

Eine View lässt sich analog wie ein Controller hinzufügen. Vorher muss jedoch ein Unterordner in dem Ordner Views mit dem entsprechenden Controllernamen, hier `Home`, erstellt werden. Über einen Rechtsklick auf den Ordner `Views` > `Hinzufügen` > `Neuer Ordner`, lässt ich dies bewerkstelligen. Daraufhin wird über Rechtsklick auf den Ordner `Home` > `Hinzufügen` > `Neues Element`, eine MVC View Page mit Namen `Index` hinzugefügt. Die erstellte View ist vorerst inhaltslos. Die Razor-Syntax ist bereits zu erkennen (siehe 7.3 Die Syntax von Razor, Seite 14).[40]

Index.cshtml

```
@*
    For more information on enabling MVC for empty projects,
    visit http://go.microsoft.com/fwlink/?LinkID=397860
*@
@{
}
```

Quellcode 8.8.1 Inhalt der View `Index`

Die Webanwendung ist ab dem jetzigen Zeitpunkt lauffähig. Es wird im Browser eine leere Ausgabe erzeugt.

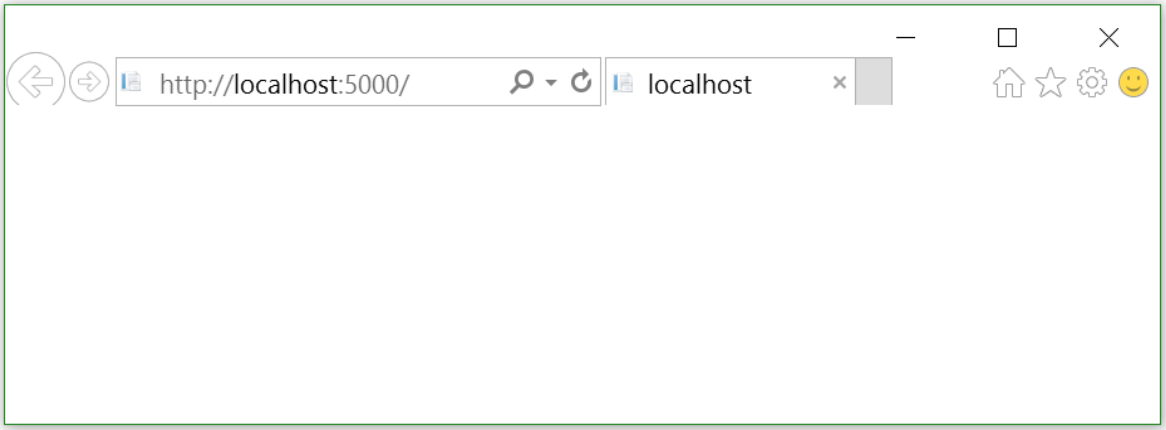


Abbildung 8.8.1 Leere Browserausgabe der View `Index`

Durch Ergänzen von Quellcode im `HomeController` und der View `Index`, werden Daten vom Controller an die View übergeben.

---

[39] Vgl. (Microsoft ASP.NET, 2016), Adding a controller
[40] Vgl. (Microsoft ASP.NET, 2016), Adding a view

```
public IActionResult Index()
{
    ViewBag.Greeting = "Hello World!";
    return View();
}
```

Quellcode 8.8.2 Ergänzungen am `HomeController`

```
@{
    <p>@ViewBag.Greeting</p>
}
```

Quellcode 8.8.3 Anpassungen an der View `Index`

Der Browser stellt Folgendes dar:

Abbildung 8.8.2 Browserausgabe mit Daten vom `HomeController`

Bei ASP.NET 5 MVC 6 RC1 Web-Apps ist die Syntax[41] des Aufrufes der Controller und deren Views immer gleich aufgebaut. ID‘s können notwendig oder optional auftreten.

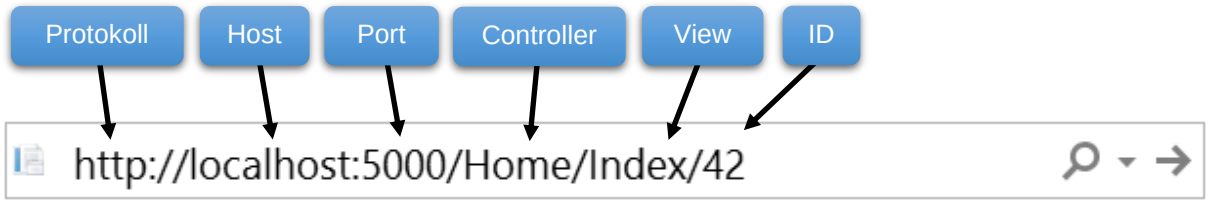


Abbildung 8.8.3 Aufbau Aufruf einer ASP.NET 5 MVC 6 RC1 Web-App

Da in der Beispielanwendung der Standard-Controller der `HomeController` und die Standard-View die `Index` ist, können diese Parameter für die Startseite der Web-App weggelassen werden (siehe Quellcode 8.6.1 Code der `Startup-Klasse` für MVC-Funktionalität, Seite 46).

[41] Vgl. (IT Wissen, kein Datum), URL (uniform resource locator)

8.9. Model hinzufügen

Die am wenigsten, aufwendige Möglichkeit ein Model einer ASP.NET 5 MVC 6 RC1 Web-App hinzuzufügen, wird über eine Klasse etabliert. Über einen Rechtsklick auf den Ordner Models in der Ordnerstruktur des Projektes > `Hinzufügen` > `Klasse` erstellt man eine leere Model-Klasse. Über öffentliche `Getter`- und `Setter`-Methoden[42] ergänzt man die erstellte Model-Klasse. Die Methoden müssen öffentlich sein, damit per Controller auf die Methoden zugegriffen werden kann.[43]

```
public class Articles
{
    0 Verweise
    public int ID { get; set; }
    0 Verweise
    public string Bezeichnung { get; set; }
    0 Verweise
    public string Artikelnummer { get; set; }
}
```

Quellcode 8.9.1 Model-Klasse mit `Getter`- und `Setter`-Methoden

8.9.1.Entity-Framework

Um von einem Controller auf die Daten aus dem Model zugreifen zu können, verwendet ASP.NET 5 MVC 6 RC1 das Entity-Framework[44]. Das Entity-Framework wird über den NuGet-Pakete-Manager installiert (siehe 8.2.1 NuGet, Seite 32). Damit das Entity-Framework eingesetzt werden kann, ist die Methode `ConfigureServices` der `Startup`-Klasse zu erweitern (siehe 8.3 Aufbau einer ASP.NET 5 MVC 6 RC1 Web-App, Seite 34).

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    var connection = @"Server=(localdb)\mssqllocaldb;Database=THBA_4DB;Trusted_Connection=True;";
    services.AddDbContext<ApplicationDbContext>(options => options.UseSqlServer(connection));

    services.AddMvc();
}
```

Quellcode 8.9.2 Anpassungen der Methode `ConfigureServices`

Es wird über einen Datentyp Variant (`var`) eine Verbindungszeichenfolge (`connectionstring`) zu einer Datenbank gespeichert. Hierbei ist es irrelevant, welche Datenbanktechnologie eingesetzt wird. Diese Verbindungszeichenfolge[45] (`connectionstring`) wird genutzt, um einen Datenbankcontext-Service hinzuzufügen. Dafür kommt die Methode `AddDbContext` des Interfaces `services` zum Einsatz. Da ASP.NET 5 MVC 6 RC1 Webanwendungen einige Services nutzen könnten, würde die `ConfigureServices`-Methode schnell unübersichtlich werden. Aus diesem Grund

---

[42] Vgl. (Microsoft Developer Network, 2016), Verwenden von Eigenschaften
[43] Vgl. (Microsoft ASP.NET, 2016), Adding a Model
[44] Vgl. (Microsoft Entity Framework, 2016)
[45] Siehe (ConnectionStrings.com, 2014)

können Verbindungszeichenfolgen (`connectionstrings`) in eine separate Konfigurationsdatei ausgegliedert werden. Diese Konfigurationsdatei heißt `appsettings.json` und wird über einen Rechtsklick auf das Projekt > `Hinzufügen` > `Neues Element` der Webanwendung hinzugefügt.

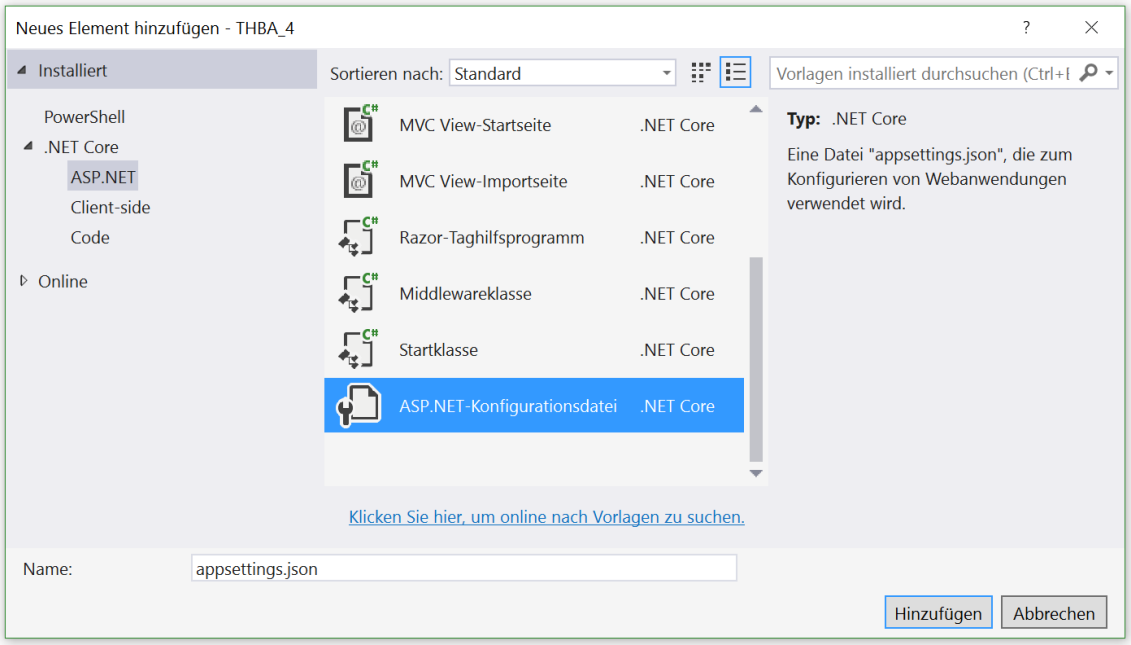


Abbildung 8.9.1 Hinzufügen der Konfigurationsdatei

Der Aufbau der Projektmappe ist wie folgt:

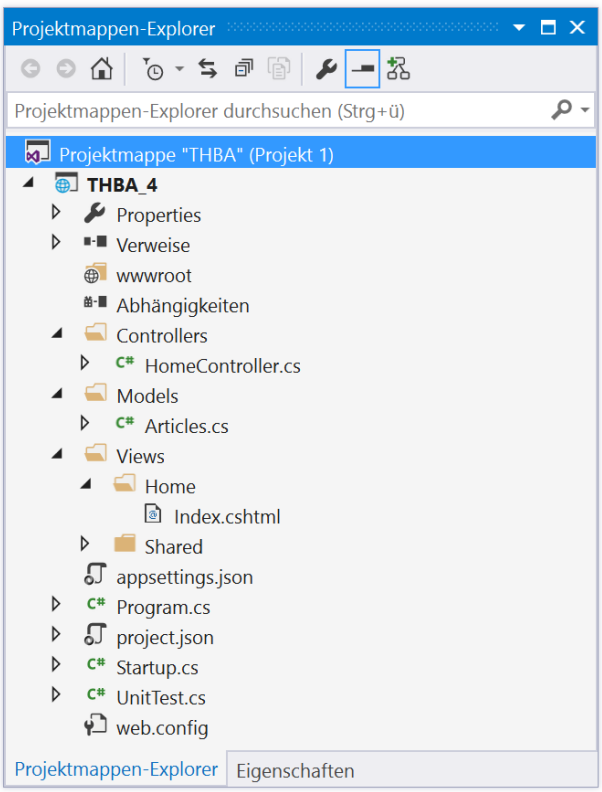


Abbildung 8.9.2 Aufbau Projekt mit Konfigurationsdatei `appsettings.json`

Verbindungszeichenfolgen (`connectionstrings`) werden in der Konfigurationsdatei `appsettings.json` wie folgt eingetragen:

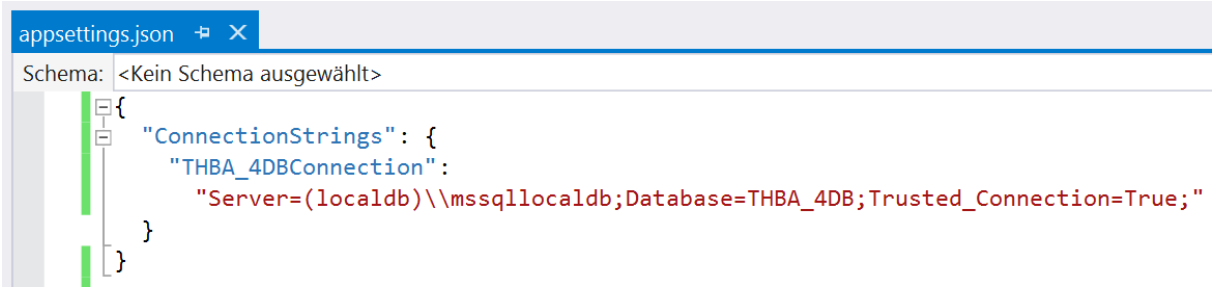


Abbildung 8.9.3 Zeichenfolge ausgelagert in `appsettings.json`

Die `ConfigureServices` Methoden der `Startup`-Klasse wird nach der Ausgliederung wie folgt angepasst:

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options => options.
        UseSqlServer(Configuration.GetConnectionString("THBA_4DBConnection")));

    services.AddMvc();
}
```

Quellcode 8.9.3 Anpassung der Methode `ConfigureServices`

Über die Methode `Configuration.GetConnectionString` ist der Zugriff auf die Verbindungszeichenfolgen (`connectionstrings`) der Konfigurationsdatei `appsettings.json` gewährleistet. Die Verbindung zu einer Datenbank ist damit vorhanden.

Zwei Ansätze zum Integrieren von Datenbanken per Entity-Framework werden unterstützt.

8.9.2. Code-First

Der Ansatz Code-First implementiert zuerst das Model und darauf basierend die Datenbank. Nachdem ein Model implementiert wurde, ist es möglich, dass eine Datenbank über die Pakete-Manager-Konsole erstellt wird. Zwei Befehle sind dafür notwendig. Zum einem wird mithilfe von `Add-Migration` das Gerüst der Datenbank erstellt. Mit `Update-Database` können Änderungen am Inhalt der Datenbank durchgeführt werden. Wird beispielsweise eine Tabelle in dem Model ergänzt, muss zuerst der Befehl `Add-Migration` ausgeführt werden. Sollen dagegen nur Daten in der Datenbank manipuliert werden, reicht `Update-Database` aus.

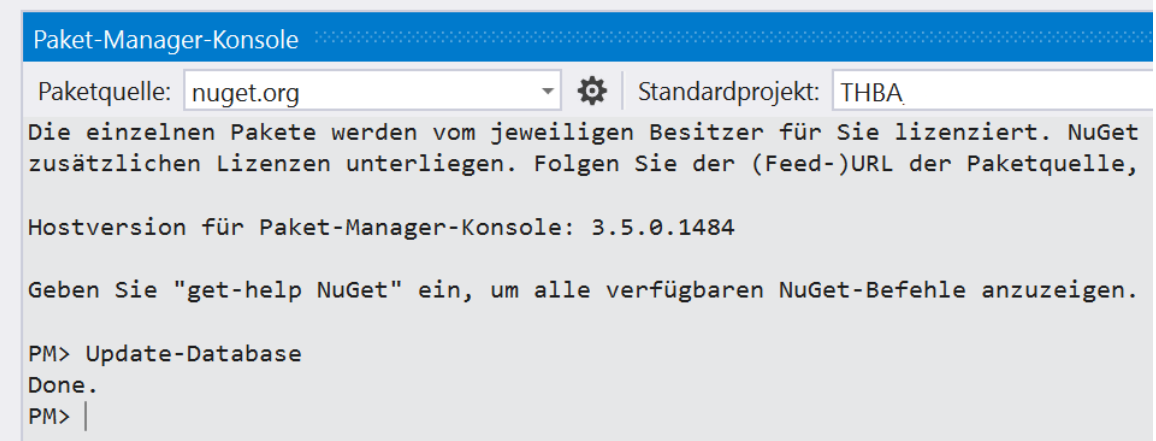


Abbildung 8.9.4 Update der Datenbank mit Pakete-Manager-Konsole

8.9.3. Database-First

Im Gegensatz zu Code-First, ist die Datenbank beim Ansatz Database-First bereits vorhanden. Hierbei wird über die Pakete-Manager-Konsole ein Model aus der gegebenen Datenbank erzeugt. Der Konsolen-Befehl hierfür lautet `Scaffold-DbContext`. Als Parameter werden die Verbindungszeichenfolge (`connectionstring`)

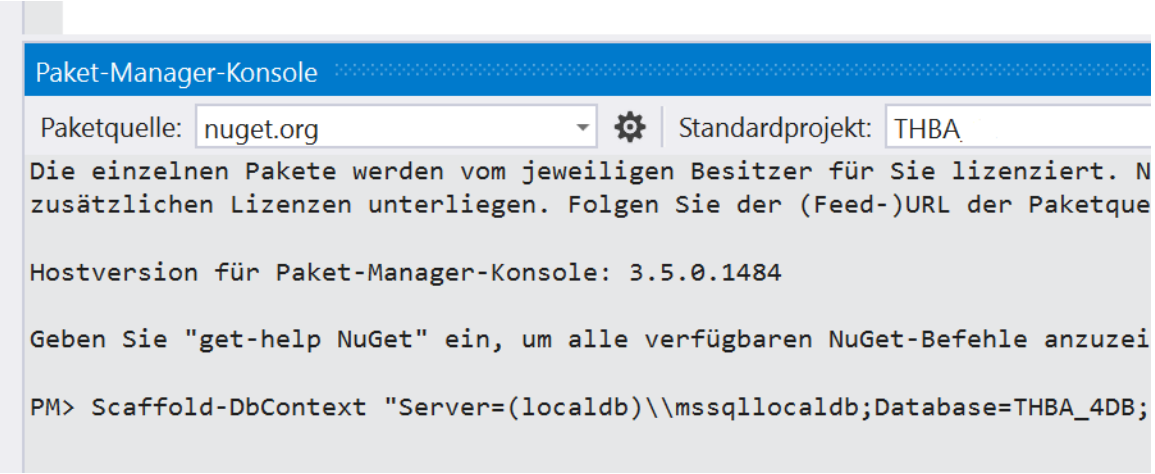


Abbildung 8.9.5 Erzeugung des Models mit Pakete-Manager-Konsole

8.10. Zusammenarbeit der MVC-Komponenten

In diesem Abschnitt wird gezeigt, wie die erstellten MVC-Komponenten zusammenarbeiten und in der View, mithilfe von Razor, über den Controller die Daten aus dem Model abfragen und manipulieren. Zu Beginn wird ein Controller implementiert, der die Daten in dem Model erstellt (`create`), Daten abruft (`read`), Daten aktualisiert (`update`) und Daten aus dem Model entfernt (`delete`) (siehe 4.2.1 Model, Seite 10). Daraufhin werden Views für die vier Vorgänge erstellt. Für das Erstellen des Controllers bietet Visual Studio Enterprise 2015 drei Vorlagen. Eine Vorlage ist für die Erstellung eines leeren Controllers. Der Controller beinhaltet keinen Quellcode und muss manuell programmiert werden. Die zweite Vorlage für einen Controller mit Lese- und Schreibmethoden implementiert die vier Operationen, um das Model zu manipulieren, ohne eine Anbindung an das Model zur Verfügung zu stellen. Dies muss durch einen Entwickler

programmiert werden. Die dritte Vorlage erstellt einen Controller basierend auf dem Model des Entity-Frameworks.

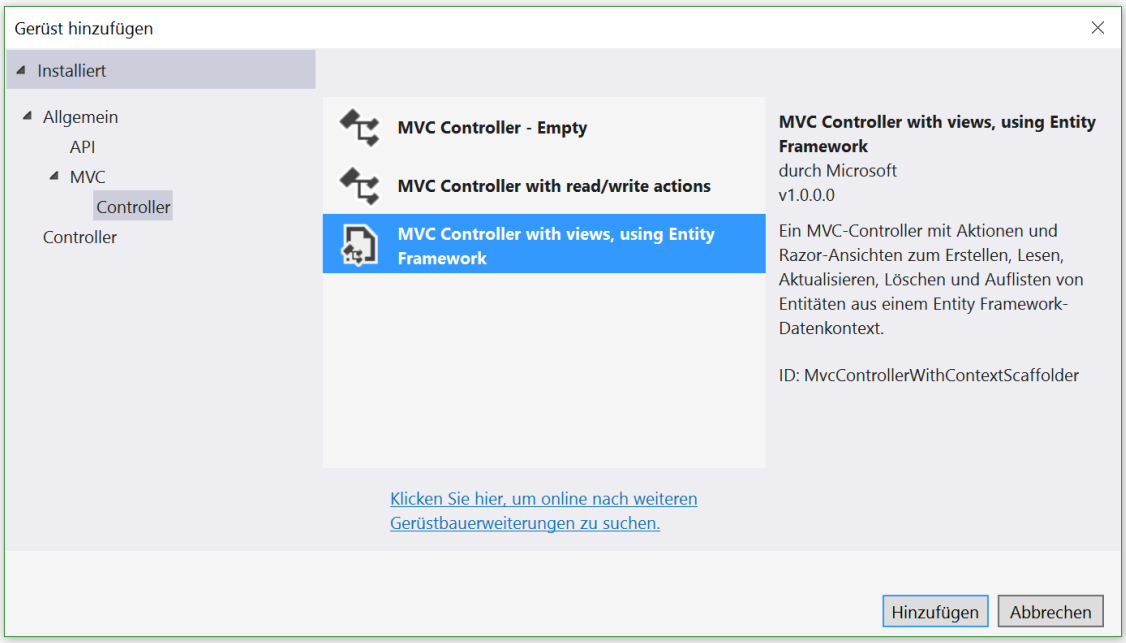


Abbildung 8.10.1 Vorlage für Erstellung des Controllers

Über Rechtsklick auf den Ordner `Controller` > `Hinzufügen` > `Controller` lässt sich der auf dem Entity-Framework-basierender Controller erstellen. Ist der Haken bei `Ansichten generiert` gesetzt, werden zudem automatisch die Views für die vier Operationen generiert.

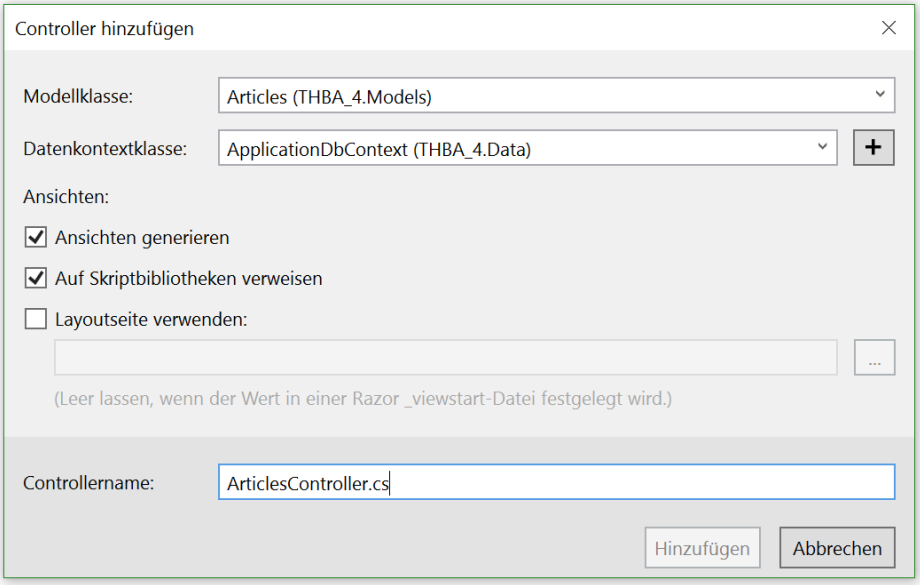


Abbildung 8.10.2 Generierung von Controller mit Views

Gleichgültig welche Vorlage genutzt wird, sind die nach Abschluss der Programmierarbeiten vorhandenen Controller und Views ähnlich. Der Controller beinhaltet die Methoden für die Datenbankoperationen und die Views stellen die Benutzeroberfläche zur Verfügung, mit der der Anwender interagiert.

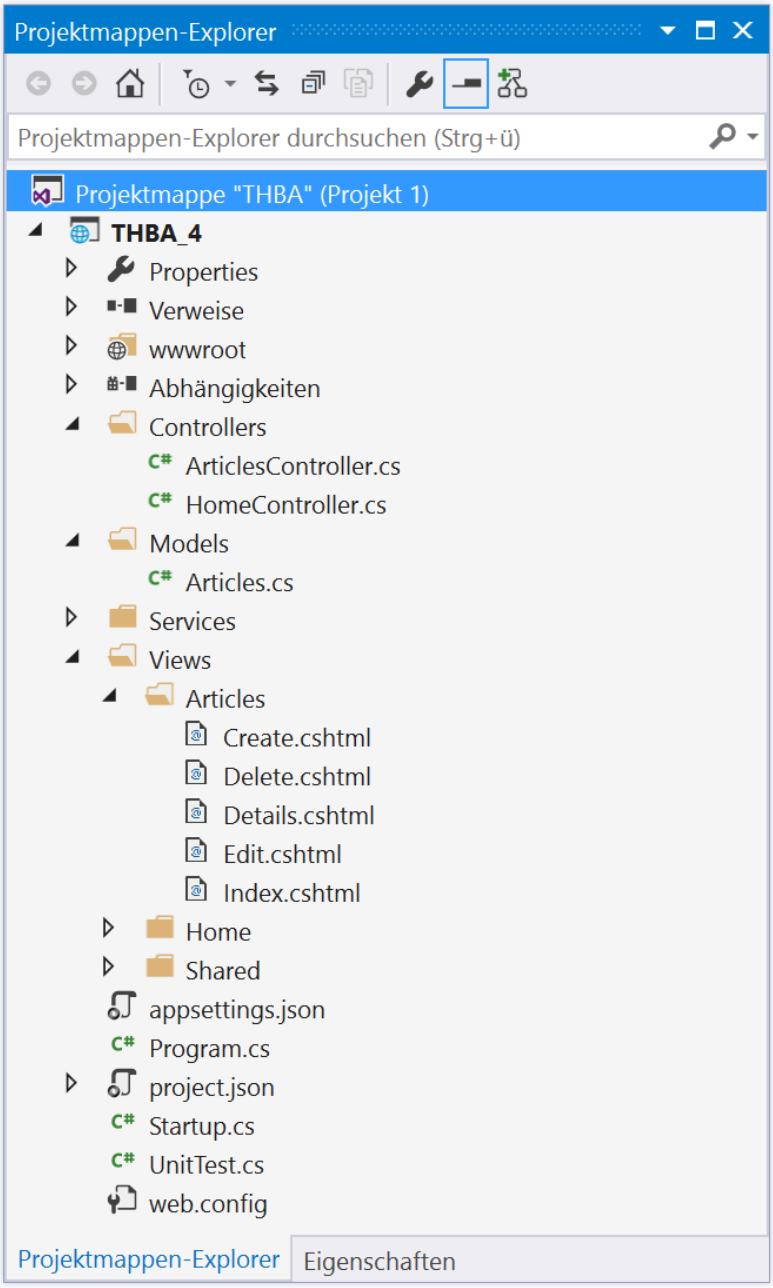


Abbildung 8.10.3 Projektmappe nach Erstellung Controller und Views

Die Controller-Klasse `ArticlesController` und die Views `Create`, `Delete`, `Details`, `Edit` und `Index` wurden erstellt. Die View `Details` zeigt weitere Informationen der einzelnen Einträge im Model. Für jede dieser Views wurde eine Methode erstellt, die jeweilige View zurückgibt.

Über die Methode `Create` wird eine neue Instanz des Models in der Datenbank erstellt.

```
// POST: Articles/Create
[HttpPost]
0 Verweise
public async Task<IActionResult> Create(
    [Bind("ID,Artikelnummer,Bezeichnung")] Articles articles)
{
    if (ModelState.IsValid)
    {
        _context.Add(articles);
        await _context.SaveChangesAsync();
        return RedirectToAction("Index");
    }
    return View(articles);
}
```

Quellcode 8.10.1 Methode `Create` des `ArticlesController`

Die Methode `ModelState.IsValid` prüft, ob die eingegeben Daten gültig sind. Ist dies der Fall, wird ein asynchroner Task gestartet, welcher die Daten in der Datenbank speichert. Daraufhin wird der Benutzer auf die `Index`-View umgeleitet. Die `Index`-Methode listet die einzelnen Einträge aus dem Model auf. Dies geschieht ebenfalls asynchron.

```
// GET: Articles
0 Verweise
public async Task<IActionResult> Index()
{
    return View(await _context.Articles.ToListAsync());
}
```

Quellcode 8.10.2 Methode `Index` des `ArticlesController`

Mithilfe der `Edit`-Methode können Daten des Models aktualisiert werden.

```
// POST: Articles/Edit/5
[HttpPost]
0 Verweise
public async Task<IActionResult> Edit(int id,
    [Bind("ID,Artikelnummer,Bezeichnung")] Articles articles)
{
    if (id != articles.ID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(articles);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ArticlesExists(articles.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction("Index");
    }
    return View(articles);
}
```

Quellcode 8.10.3 Methode `Edit` des `ArticlesController`

Zu Beginn wird überprüft, ob die übergebene ID der Modelinstanz eine gültige Modelinstanz repräsentiert. Ist dies gegeben, wird, wie bei der `Create`-Methode, auf valide Daten überprüft. Anschließend wird versucht die Daten in der Datenbank zu aktualisieren. Sollte in der Zeit zwischen dem Aufruf der View und dem Aktualisieren der Daten eine Manipulation des Models stattgefunden haben, wird eine Ausnahme (`Exception`) geworfen. Sollte die Modelinstanz gelöscht worden sein, wird ein http-Fehler 404 zurückgegeben. Dieser besagt, dass die angeforderte Ressource nicht gefunden werden konnte. Nach erfolgreichem Abschluss der Aktionen, wird auf die `Index`-View umgeleitet.

Daten aus dem Model werden über die `Delete`-Methode entfernt. Der Vorgang besteht aus zwei Teilen, da bei `Delete`-Anforderungen eine Bestätigung des Benutzers angefordert wird.

```
// GET: Articles/Delete/5
0 Verweise
public async Task<IActionResult> Delete(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var articles = await _context.Articles.SingleOrDefaultAsync(m => m.ID == id);
    if (articles == null)
    {
        return NotFound();
    }

    return View(articles);
}

// POST: Articles/Delete/5
[HttpPost, ActionName("Delete")]
0 Verweise
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var articles = await _context.Articles.SingleOrDefaultAsync(m => m.ID == id);
    _context.Articles.Remove(articles);
    await _context.SaveChangesAsync();
    return RedirectToAction("Index");
}
```

Quellcode 8.10.4 Methode `Delete` des `ArticlesController`

Wird keine ID oder eine ID übergeben, die im Model nicht vorhanden ist, wird ein http-Fehler 404 zurückgegeben. Sollte eine Modelinstanz gefunden werden, wird der Benutzer gefragt, ob die Modelinstanz wirklich gelöscht werden soll. Bestätigt der Benutzer die Löschung, wird der Eintrag aus der Datenbank entfernt und auf die `Index`-View umgeleitet.

Die zu den Methoden gehörenden Views werden nun beschrieben, beginnend mit der View `Create`.

```
@model THBA_4.Models.Articles

@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Create</title>
</head>
<body>
    <form asp-action="Create">
        <label asp-for="Artikelnummer"></label>
        <input asp-for="Artikelnummer" />
        <label asp-for="Bezeichnung"></label>
        <input asp-for="Bezeichnung" />
        <input type="submit" value="Create" />
    </form>
    <div>
        <a asp-action="Index">Back to List</a>
    </div>
</body>
</html>
```

Quellcode 8.10.5 Frontendcode der `Create`-View

Zu Beginn des Dokumentes erkennt man die Übergabe eines Models vom Typ `Articles`. Dies geschieht per Razor mit dem Kennwort `model`. Der Razor-Abschnitt darunter teilt dem Compiler mit, dass keine Layout-View verwendet wird (siehe 8.12 Responsives Webdesign, Seite 66). Da es sich um eine http-Post-Methode handelt, was bedeutet, dass Daten vom Browser an den http-Server gesendet werden, muss ein `form`-Element verwendet werden. In diesem sind die Beschriftungen (`label`) und die einzelnen Felder (`input`) für die Eingaben der Daten vorhanden. Eine Schaltfläche wird für das Absenden[46] (`post`) der Daten benutzt.

[46] Siehe (Tecchannel, 2001), HTTP-Grundlagen - Hypertext Transfer Protocol

Wird der Controller `ArticesController` und die dort enthaltene Methode `Create` aufgerufen, ist die Browserausgabe der View `Create` wie folgt:

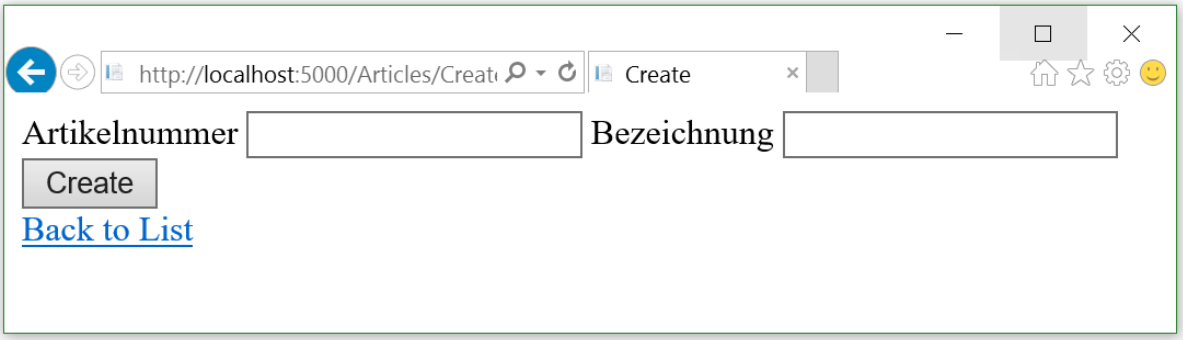


Abbildung 8.10.4 Ausgabe der `Create`-View

Über diese View kann eine Instanz des Models angelegt werden. Sind Daten vorhanden, werden diese in der View `Index` gelistet. Die View Index ist wie folgt aufgebaut:

```
@model IEnumerable<THBA_4.Models.Articles>

<!DOCTYPE html>
<html>
<head>...</head>
<body>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Artikelnummer)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Bezeichnung)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Artikelnummer)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Bezeichnung)
            </td>
            <td>
                <a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |
                <a asp-action="Details" asp-route-id="@item.ID">Details</a> |
                <a asp-action="Delete" asp-route-id="@item.ID">Delete</a>
            </td>
        </tr>
}
```

Quellcode 8.10.6 Code-Snippet der `Index`-View

Per Razor-Kennwort `model` wird eine Liste vom Typ des Models an die View übergeben. Das Kennwort `Html` bietet einzelne Methoden, die vom Compiler in HTML-Elemente zur Laufzeit umgewandelt werden. Beispielsweise wird aus `Html.DisplayNameFor` simpler Text. Aus `Html-DisplayFor` wird ein Eingabefeld vom Typ Input. Die `foreach`-Schleife (siehe 7.13 foreach-Schleife, Seite 25) listet alle Elemente der Datenbank untereinander auf. Dies geschieht ebenfalls per Razor.

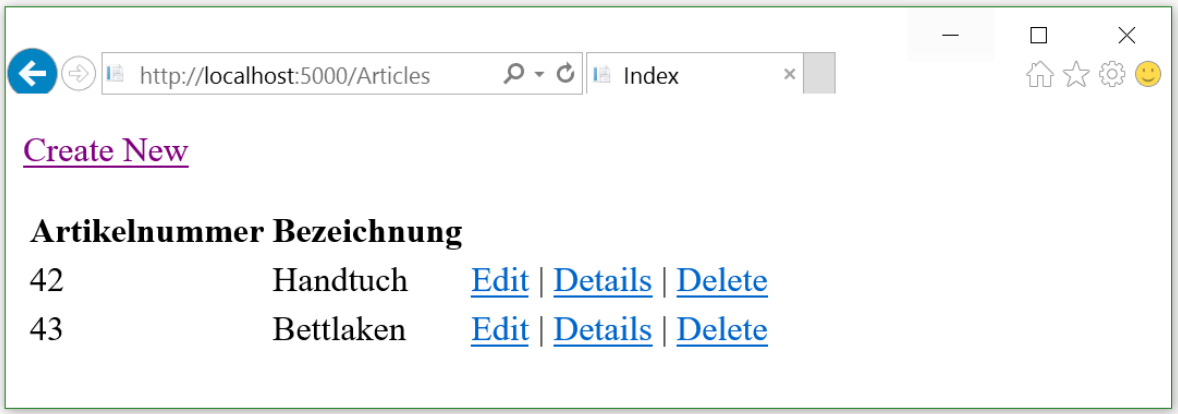


Abbildung 8.10.5 Ausgabe der `Index`-View

Sind Daten vorhanden, können diese bearbeitet werden. Dazu wird die View `Edit` mit der `ID` der Modelinstanz aufgerufen.

```
@model THBA_4.Models.Articles

<!DOCTYPE html>
<html>
<head>...</head>
<body>
    <form asp-action="Edit">
        <label asp-for="Artikelnummer" ></label>
        <input asp-for="Artikelnummer" />
        <label asp-for="Bezeichnung" ></label>
        <input asp-for="Bezeichnung" />
    </form>
    <div>
        <a asp-action="Index">Back to List</a>
    </div>
</body>
</html>
```

Quellcode 8.10.7 Inhalt der `Edit`-View

Mithilfe von Razor wird wieder eine Instanz des Models an die View übergeben. Bezeichnungsfelder und Eingabefelder werden kompiliert.

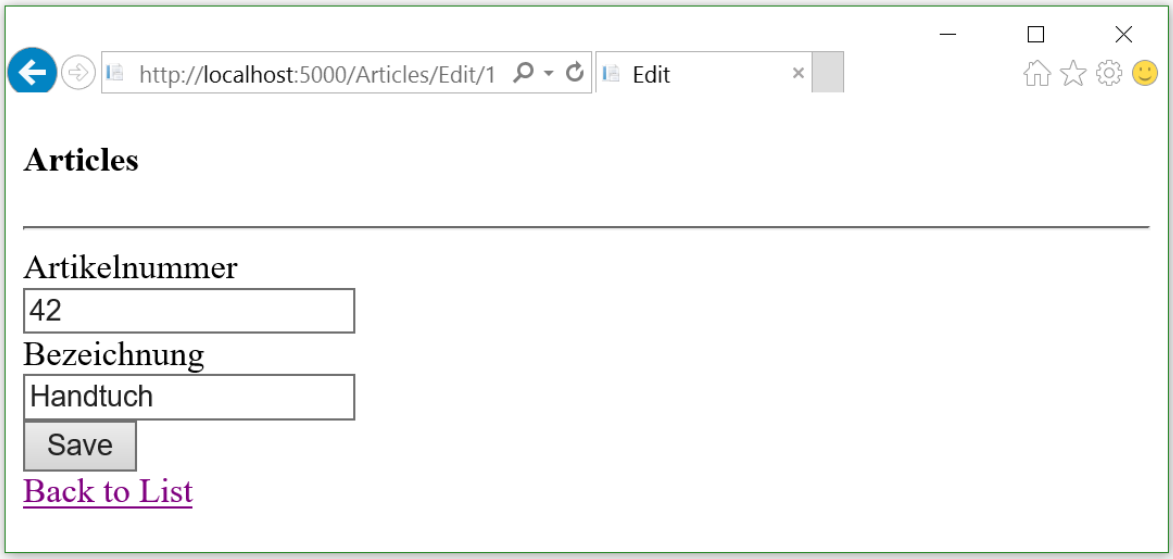


Abbildung 8.10.6 Ausgabe der `Create`-View

In der `Edit`-View können Änderungen an der Modelinstanz durchgeführt werden. Über die `Delete`-View werden Daten aus der Datenbank gelöscht.

```
@model THBA_4.Models.Articles

<!DOCTYPE html>
<html>
<head>...</head>
<body>
<h3>Are you sure you want to delete this?</h3>

            @Html.DisplayNameFor(model => model.Artikelnummer)

            @Html.DisplayFor(model => model.Artikelnummer)

            @Html.DisplayNameFor(model => model.Bezeichnung)

            @Html.DisplayFor(model => model.Bezeichnung)

    <form asp-action="Delete">
        <div>
            <input type="submit" value="Delete" /> |
            <a asp-action="Index">Back to List</a>
        </div>
    </form>
</body>
</html>
```

Quellcode 8.10.8 Inhalt der `Delete`-View

Die Daten aus dem Model werden analog zu der `Edit`-View dargestellt.

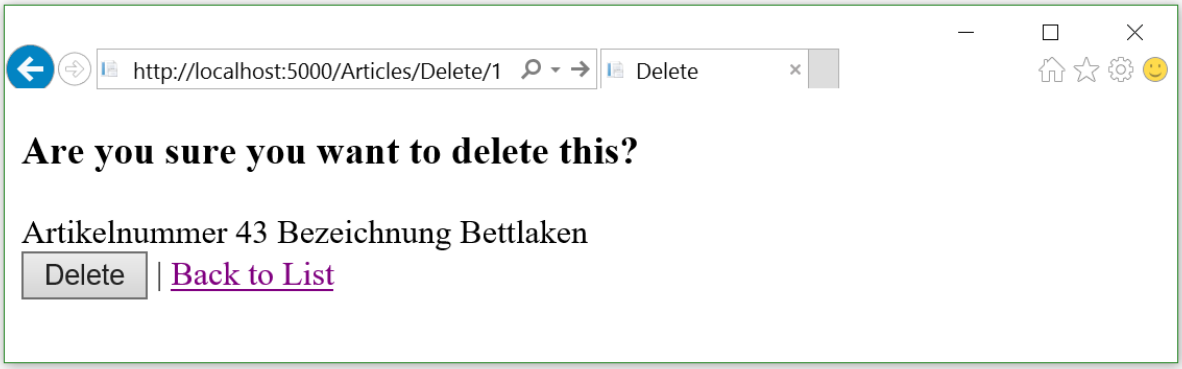


Abbildung 8.10.7 Ausgabe der `Delete`-View

Auf die Methode des Controllers und die View für die Details, wird hier nicht weiter eingegangen. Sie wird nur vollständigkeitshalber erwähnt. Das Erläutern des Zusammenhanges der MVC-Komponenten ist abgeschlossen. Jegliche Vorgänge für das Interagieren mit Views und die Steuerung mithilfe des Controllers wurden gezeigt.

8.11. Authentifizierung

Für die Benutzerauthentifizierung wird bei ASP.NET 5 MVC 6 RC1 Webanwendungen die Bibliothek `ASP.NET Core Identity` genutzt. Diese Bibliothek wird über den NuGet-Pakete-Manager installiert (siehe 8.2.1 NuGet, Seite 32). Um ASP.Net Core Identity in einem Projekt nutzen zu können, ist es notwendig die vorhandene `ConfigureServices`-Methode der `Startup`-Klasse zu ergänzen.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("THBA_4DBConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();
}
```

Quellcode 8.11.1 `Startup`-Klasse mit `Identity`

Über die Methode `AddIdentity` wird die Identitätsverwaltung als Dienst der Anwendung registriert. Die Benutzerinformationen und Logindaten werden verschlüsselt in der Datenbank mithilfe des Entity-Frameworks verwaltet. Die Methode `AddEntityFrameworkStores` implementiert den Datenbankkontext, der die Ansteuerung der Datenbank übernimmt. `AddDefaultTokenProviders` stellt Funktionalitäten für das Zurücksetzen von Logininformationen der Benutzer bereit.

Des Weiteren ist es notwendig, dass die `Configure`-Methode der Startup-Klasse der ASP.NET 5 MVC 6 RC1 Web-App mitteilt, dass die ASP.NET Core Identity genutzt werden soll.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory)
{
    app.UseIdentity();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Quellcode 8.11.2 Code der `Configure`-Methode mit `Identity`

Mithilfe der Methode `UseIdentity` wird der Webanwendung mitgeteilt, dass `Identity` genutzt werden soll. Um eine Benutzerverwaltung zu implementieren, wird ein Controller `AccountController` ergänzt (siehe 8.7 Controller hinzufügen, Seite 47). Über den Controller werden die Vorgänge Benutzer registrieren, Benutzer anmelden und Benutzer abmelden abgebildet. Der Controller kann um weitere Vorgänge, wie Kennwort vergessen, Kennwort senden und Kennwort zurücksetzen, erweitert werden.

```
[Authorize]
4 Verweise | jonnyodt, Vor 22 Stunden | 1 Autor, 1 Änderung
public class AccountController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;
```

Quellcode 8.11.3 Codesnippet Instanzen des `AccountControllers`

Eine Instanz der Klasse `UserManager` wird benötigt um einen neuen Benutzer zu registrieren. Die Instanz der `SignInManger`-Klasse wird für den An- und Abmeldevorgang verwendet.

```
// POST: /Account/Register
[HttpPost]
[AllowAnonymous]
0 Verweise | jonnyodt, Vor 22 Stunden | 1 Autor, 1 Änderung
public async Task<IActionResult> Register(RegisterViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
        var result = await _userManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            return RedirectToLocal(returnUrl);
        }
        AddErrors(result);
    }
    return View(model);
}
```

Quellcode 8.11.4 `Register`-Methode des `AccountControllers`

Der Register-Methode wird eine Modelinstanz vom Typ `RegisterViewModel` übergeben. Diese Instanz beinhaltet die Emailadresse und das Kennwort des Benutzers. Mithilfe der Methode `CreateAsync` wird ein Benutzer in der Datenbank angelegt. Die Methode `Login` nutzt die Instanz des `SignInManagers`, um den Benutzer über die Methode `PasswortSignInAsync` anzumelden.

```
// POST: /Account/Login
[HttpPost]
[AllowAnonymous]
2 Verweise | jonnyodt, Vor 22 Stunden | 1 Autor, 1 Änderung
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
{
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(model.Email,
            model.Password, model.RememberMe, lockoutOnFailure: false);
        if (result.Succeeded)
        {
            return RedirectToLocal(returnUrl);
        }
        else
        {
            ModelState.AddModelError(string.Empty, "Invalid login attempt.");
            return View(model);
        }
    }
    return View(model);
}
```

Quellcode 8.11.5 `Login`-Methode des `AccountControllers`

Die Methode `Logout` meldet den Benutzer mithilfe des `SignInManagers` und der Methode `SignOutAsync` vom System ab.

```
// POST: /Account/LogOff
[HttpPost]
0 Verweise | jonnyodt, Vor 22 Stunden | 1 Autor, 1 Änderung
public async Task<IActionResult> LogOff()
{
    await _signInManager.SignOutAsync();
    _logger.LogInformation(4, "User logged out.");
    return RedirectToAction(nameof(HomeController.Index), "Home");
}
```

Quellcode 8.11.6 `Logout`-Methode des `AccountControllers`

Um den Zugriff auf die Funktionen der ASP.NET 5 MVC 6 RC1 Webanwendung auf Benutzerebene einzuschränken, können Methoden der Controller nach Integration von `Identity` angepasst werden. Oberhalb einer Methode des Controllers kann das Attribut `AllowAnonymous` ergänzt werden. Dies erlaubt dem Benutzer Zugriff auf die Methode ohne bereits am System angemeldet zu sein. Beispielsweise muss die `Register`-Methode des `AccountControllers` mit diesem Attribut versehen werden. Über das Attribut `Authorize` wird festgelegt, dass der Benutzer angemeldet sein muss, um die Methode des Controllers aufzurufen. `Authorize` kann ebenfalls um Benutzerrollen erweitern werden. Auf die Umsetzung wird hier nicht weitereingegangen, da hierfür eine rollenbasierte Authentifizierung[47] implementiert werden muss. Dabei wird dem Benutzer ein weiteres Attribut beim Registrierungsvorgang mitgegeben, welches die Rolle des Benutzers darstellt. [48]

8.12. Responsives Webdesign

Um Web-Apps auf möglichst vielen Endgeräte nutzbar und ansehnlich darzustellen, wird auf den Ansatz des responsiven Webdesign gesetzt. Dieser wurde von Ethan Marcotte im Jahr 2010 veröffentlicht[49]. Die Webanwendung ist dabei in der Lage den darzustellenden Quellcode an die Displaygröße und Auflösung des Endgerätes anzupassen. ASP.NET 5 MVC 6 RC1 Web-Apps haben eine direkte Möglichkeit, das responsive Webdesign-Framework Bootstrap zu implementieren. Dieses wird über den Bower-Pakete-Manager installiert (siehe 8.2.2 Bower, Seite 33). Bootstrap wurde von einem Designer und einem Entwickler der Firma Twitter ins Leben gerufen. Dieses Framework setzt auf eine Kombination aus Javascript und CSS, um die responsiven Ansätze zu implementieren. Fügt man einer ASP.NET 5 MVC 6 RC1 Web-App Bootstrap hinzu, ist es möglich Layout-Dateien zu ergänzen, die einen Container für die Views der Anwendung bereitstellen. Im Projektordner `Shared` werden die Layout-Dateien abgelegt. Über einen `Rechtsklick` auf den Ordner `Shared` > `Hinzufügen` > `Neues Element` lassen sich MVC-View-Layoutseiten hinzufügen.

[47] Siehe (Microsoft ASP.NET, 2016), Role based Authorization
[48] Vgl. (Microsoft ASP.NET, 2016), Introduction to Identity
[49] Vgl. (Marcotte, 2010), Responsive Web Design

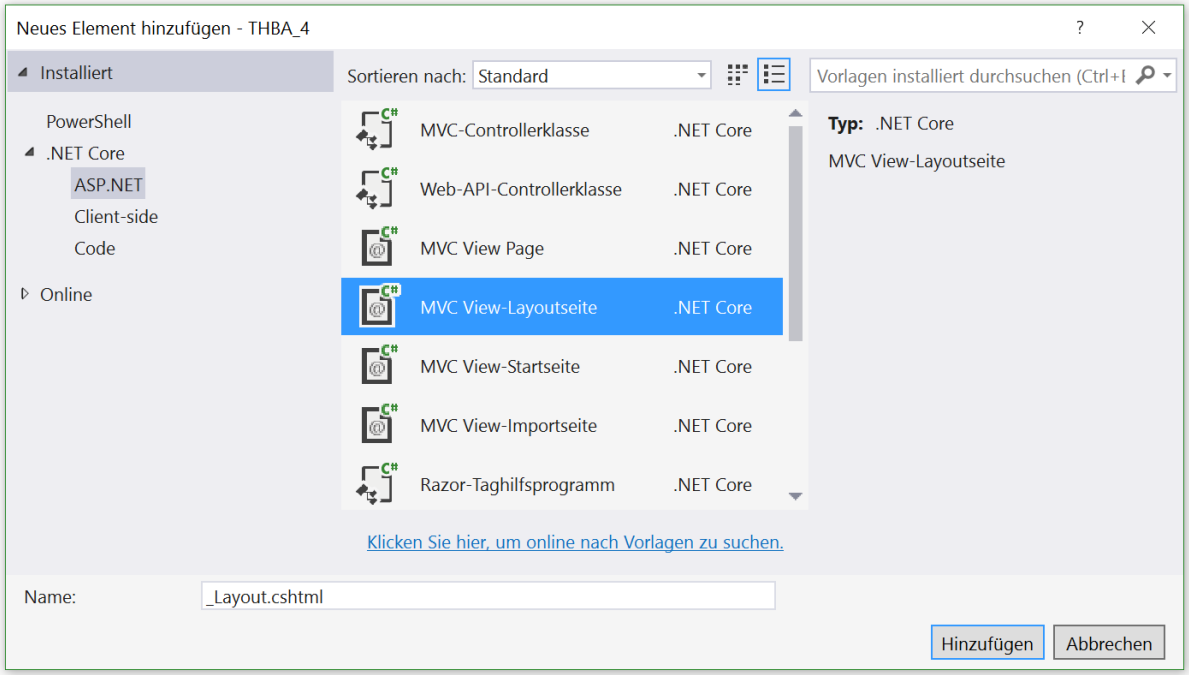


Abbildung 8.12.1 Hinzufügen einer MVC-View-Layoutseite

Der Inhalt einer Layoutseite hat den folgenden Aufbau:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - THBA_4</title>

    <environment>...</environment>
    <environment>...</environment>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">...</div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; 2016 - THBA_4</p>
        </footer>
    </div>

    <environment>...</environment>
    <environment>...</environment>

    @RenderSection("scripts", required: false)
</body>
</html>
```

Quellcode 8.12.1 Inhalt einer MVC-View-Layoutseite

Es ist zu erkennen, dass diverse CSS-Klassen in dem Quellcode der Seite vorhanden sind. Über den Razor-Befehl `Renderbody` wird der Inhalt der einzelnen Seiten der Webanwendung gerendert. Um dies zu verdeutlichen, wird ein neuer Controller `ArticlesResponsive` ergänzt (siehe 8.7 Controller hinzufügen, Seite 47).

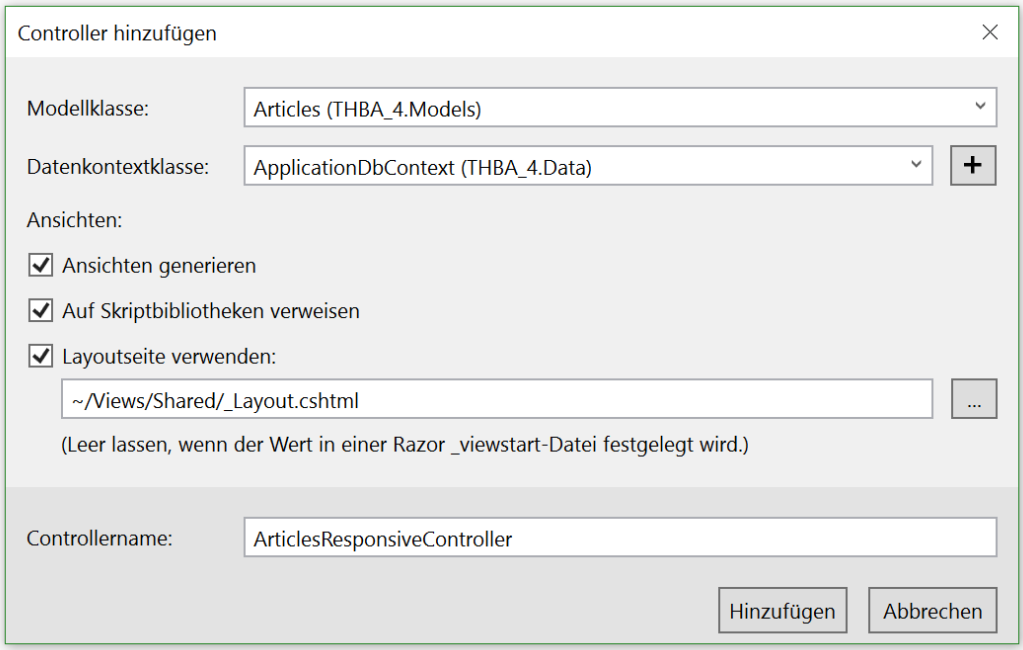


Abbildung 8.12.2 Controller mit Layoutseite hinzufügen

Wird der Haken bei `Layoutseite verwenden` gesetzt, besteht die Möglichkeit eine Layoutdatei als Vorlage auszuwählen. Die Views werden ebenfalls generiert. Nach Erstellung des Controllers ist der Quellcode der View `Index` wie folgt:

```
@model IEnumerable<THBA_4.Models.Articles>

@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Index</h2>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>...</thead>
    <tbody>
@foreach (var item in Model) {
        <tr>...</tr>
}
    </tbody>
</table>
```

Quellcode 8.12.2 Markup `Index`-View mit Layoutseite

Es ist zu erkennen, dass das Attribut `Layout` im Razor Codeblock einen Verweis auf die Layoutseite beinhaltet. Nach dem Hosten der Webanwendung, wird folgendes im Browser des jeweiligen Endgerätes dargestellt.

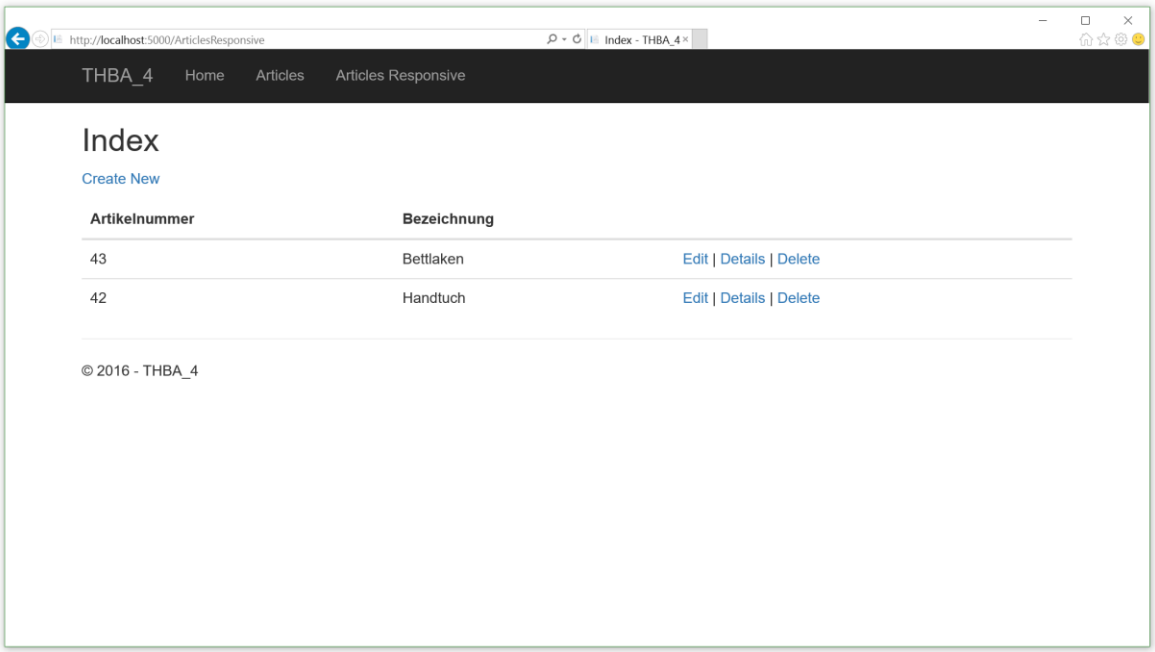


Abbildung 8.12.3 Browserausgabe Desktop-PC

Auf einem mobilen Endgerät wird die ASP.NET 5 MVC 6 RC1 Web-App, basierend auf der Displaygröße, optimal dargestellt.

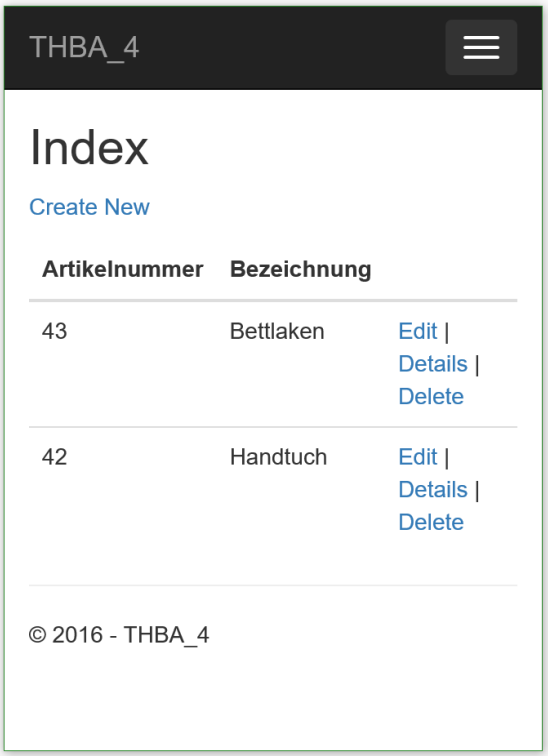


Abbildung 8.12.4 Browserausgabe auf einem mobilen Endgerät

Die ASP.NET 5 MVC 6 RC1 Web-App entspricht dem Design-Ansatz des responsiven Webdesigns. Sehr positiv ist dabei die simple Implementierung dieses Ansatzes in ein ASP.NET 5 MVC 6 RC1 Webprojekt.

8.13. Quellcodeverwaltung

Der erstellte Quellcode wird über Repositories veröffentlicht. Auf die allgemeine Funktionsweise von Repositories wird hier nicht weiter eingegangen.[50] Dies ist vor allem bei Entwicklungsprojekten mit großen Entwicklerteams sinnvoll. Zwei Repositories werden direkt aus Visual Studio Enterprise 2015 unterstützt. Zum einen die internen Visual Studio Team Services und zum anderen das Repository GitHub. Über den Team-Explorer von Visual Studio Enterpriese 2015 ist es möglich, sich mit einem oder beiden Repositories zu verbinden und die einzelnen Programmversionen und Programmänderungen zu publizieren. Der Team-Explorer wird über `Ansicht` > `Team-Explorer` geöffnet.

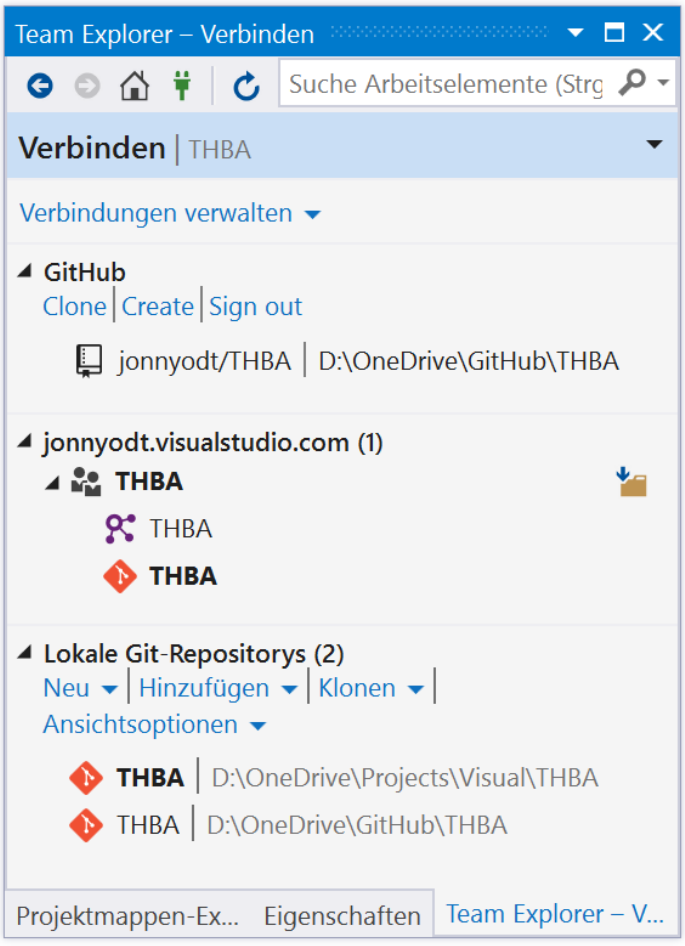


Abbildung 8.13.1 Team-Explorer mit Verbindung zu GitHub und Team Services

[50] Siehe (Schütze, 2008), Verteilte Versionsverwaltung

Folgende Möglichkeiten der Versionsverwaltung bieten die Visual Studio Team-Services:

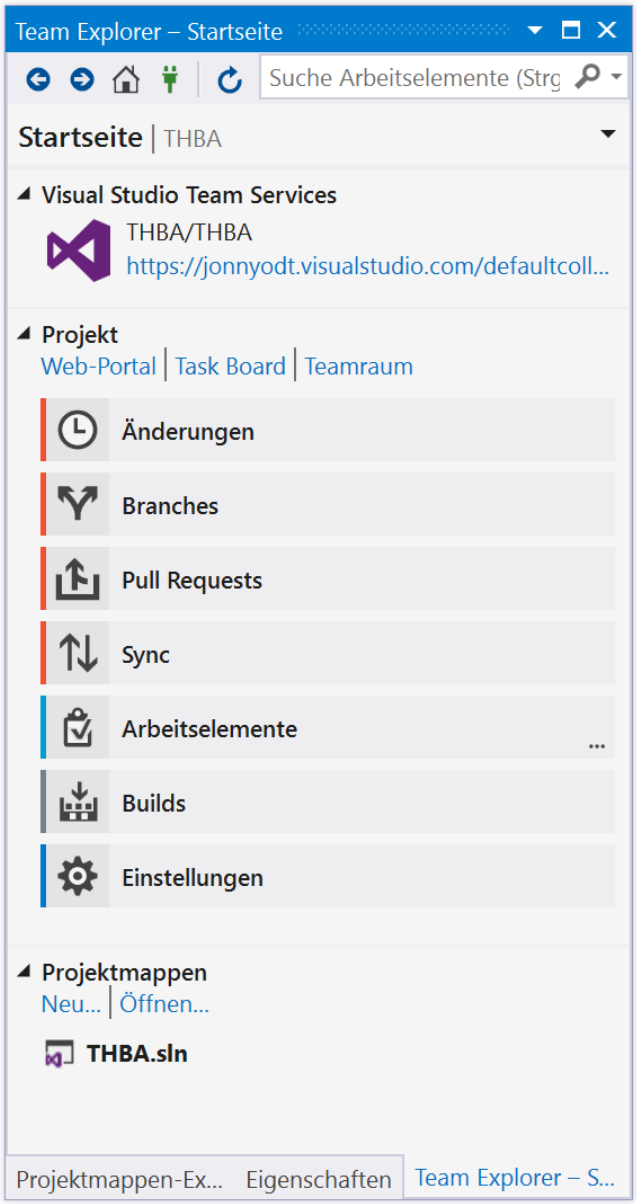


Abbildung 8.13.2 Funktionen der Visual Studio Team Services

Die Funktionen von der Anbindung an GitHub-Repositories sind wie folgt:

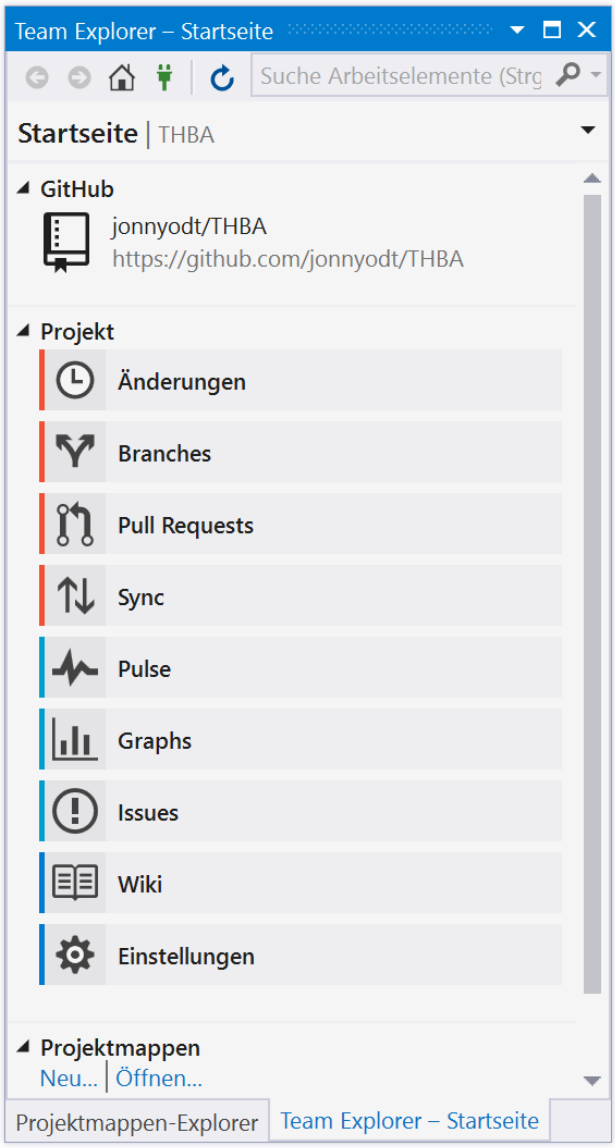


Abbildung 8.13.3 Funktionen der GitHub-Anbindung

8.14. Veröffentlichung

Der letzte Schritt bei der Entwicklung und Veröffentlichung einer ASP.NET 5 MVC 6 RC1 Web-App ist die Veröffentlichung selbst. Über die Veröffentlichung wird der Zugriff auf die Webanwendung gegeben. Mithilfe von Visual Studio Enterprise 2015 lassen sich drei unterschiedliche Möglichkeiten nutzen. Zum einen kann der `Kestrel`-Webserver genutzt werden.

8.14.1. Kestrel-Webserver[51]

Soll die Anwendung über den `Kestrel`-Webserver gehostet werden, ist ein Erstellen der Anwendung im `Release`-Modus notwendig. Im `Release`-Modus wird die finale Version der Anwendung kompiliert. Über die Auswahl im oberen Menü von Visual Studio Enterprise 2015, wird `Release` ausgewählt.

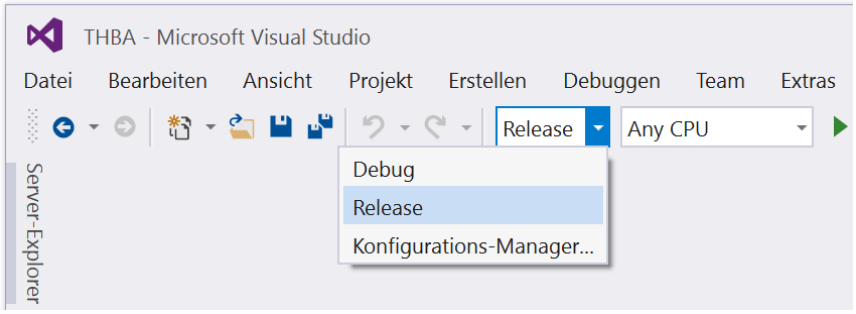
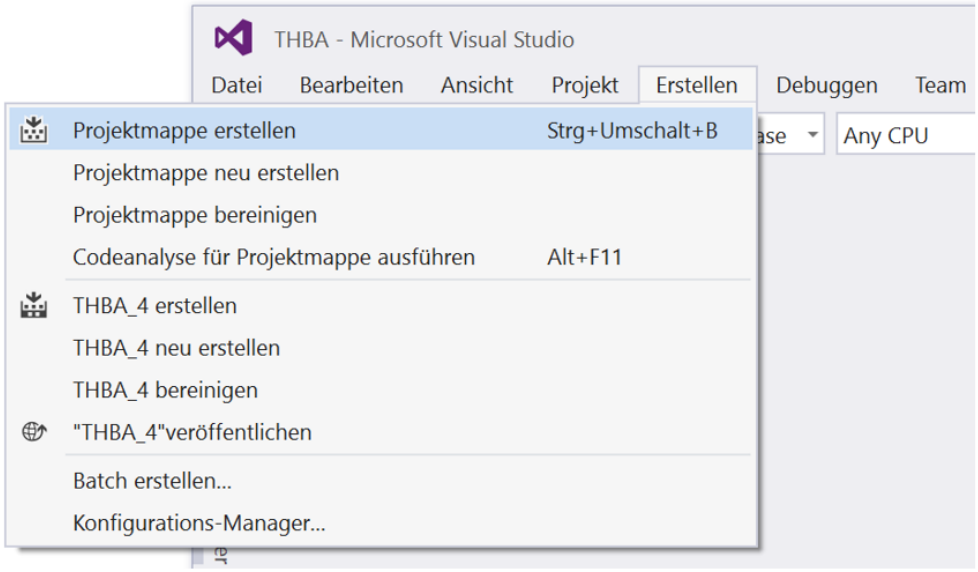


Abbildung 8.14.1 Auswahl Release-Modus

Anschließend wird das Projekt über `Erstellen` > `Projektmappe erstellen` kompiliert.



[51] Siehe (Microsoft ASP.NET, 2016), Servers

Abbildung 8.14.2 Projektmappe final erstellen

Im Projektorder befinden sich nun alle notwendigen Dateien, um die ASP.NET 5 MVC 6 RC1 Web-App zu starten. Über den Konsolenbefehl `dotnet.exe run` wird der `Kestrel`-Webserver gestartet und veröffentlicht das Projekt.

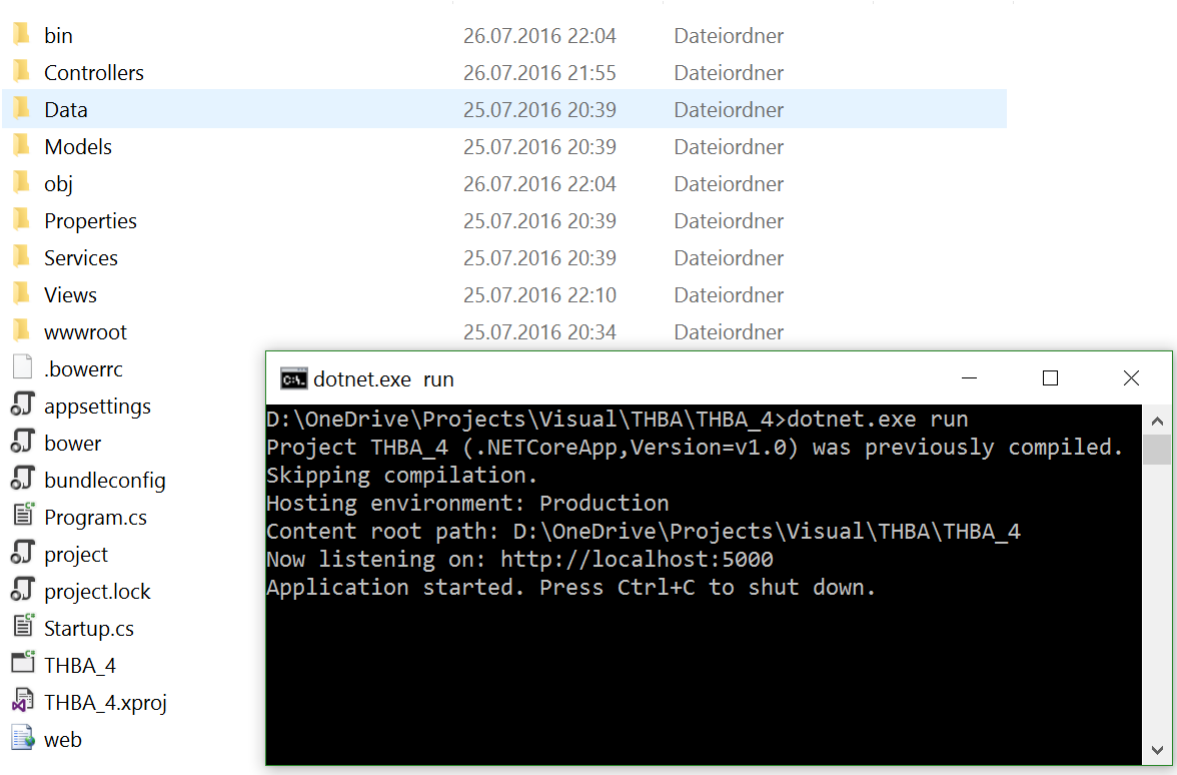


Abbildung 8.14.3 Veröffentlichung mit `Kestrel`-Webserver

Die ASP.NET 5 MVC 6 RC1 Web-App lässt sich über einen Browser aufrufen:

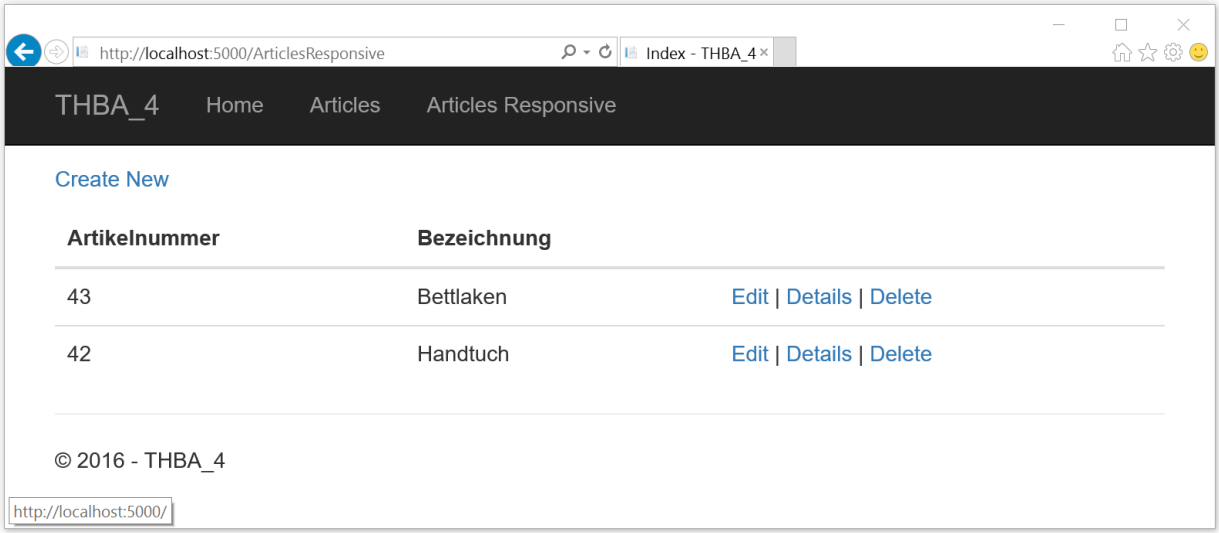


Abbildung 8.14.4 Browserausgabe basierend auf `Kestrel`-Webserver

Bei Bedarf muss die Verbindungszeichenfolge (`connectionstring`) zur Datenbank angepasst werden. Dies ist beispielsweise der Fall, wenn die Datenbank nicht lokal auf dem Gerät verfügbar ist. Da der `Kestrel`-Webserver auf den Plattformen Windows, Mac OSX und Linux lauffähig ist, lässt sich eine ASP.NET 5 MVC 6 RC1 auf fast jedem Gerät hosten. Dadurch kann die Webanwendung ebenfalls als `Docker`-Container[52] integriert werden und ist auf einem Microcomputer, wie dem `Rasperry Pi`[53], ausführbar.

8.14.2. Microsoft Internet Information Services (IIS)[54]

Die zweite Möglichkeit eine ASP.NET 5 MVC 6 RC1 Web-App zu veröffentlichen, ist über die Microsoft Internet Information Services (IIS) gegeben. Jegliche Microsoft Windows Betriebssysteme ermöglichen die Installation dieser Services.

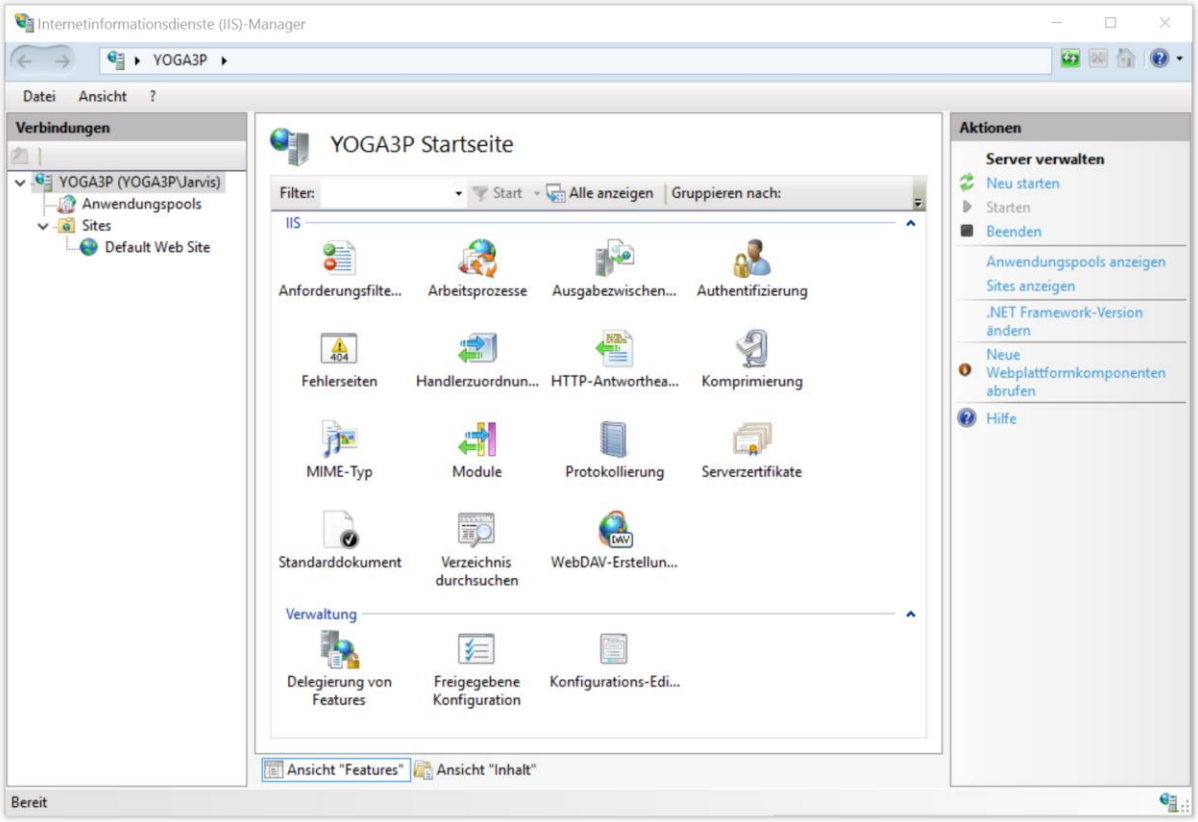


Abbildung 8.14.5 Oberfläche des IIS

Um mit Visual Studio Enterprise 2015 auf die Microsoft Internet Information Services (IIS) zugreifen zu können, ist es notwendig ein Werkzeug mit dem Namen Web Deploy[55] über den Webplattform-Installer zu installieren. Dieses muss mindestens in

[52] Siehe (Docker, 2016), What is Docker
[53] Siehe (Rasperry Pi, kein Datum), About us
[54] Siehe. (Microsoft IIS, kein Datum), Overview
[55] Vgl. (Microsoft IIS, 2011), Installing and Configuring Web Deploy on IIS 7

Version 3.5 vorliegen, damit es mit Visual Studio Enterprise 2015 kompatibel ist. Web Deploy wird über die Aktion `Neue Webplattformkomponenten abrufen` installiert.

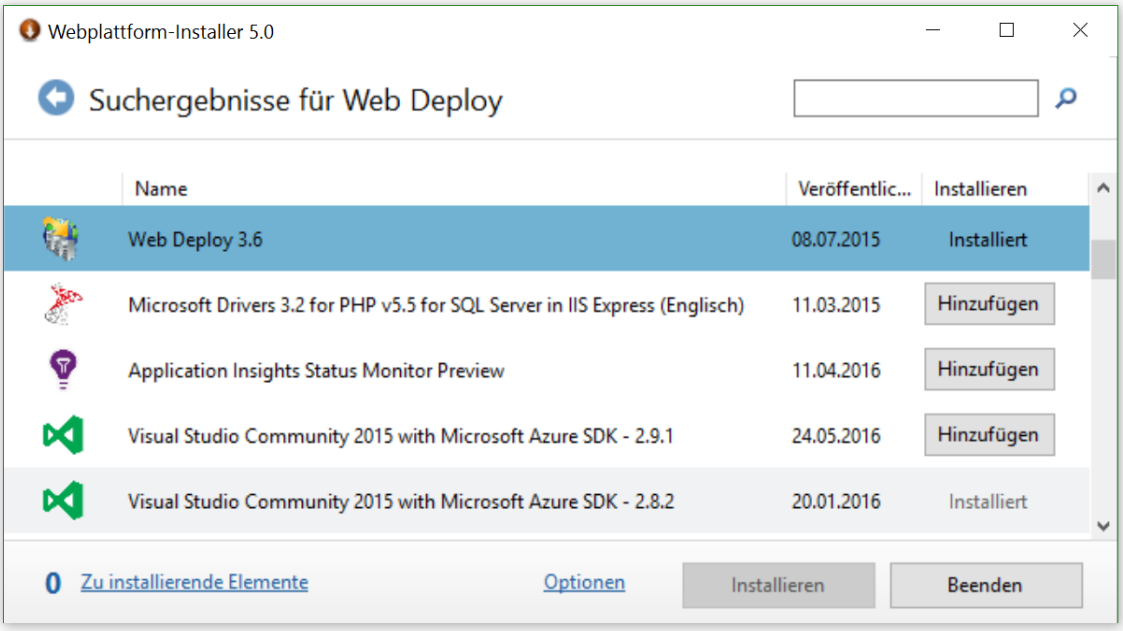


Abbildung 8.14.6 Installation Web Deploy 3.6

Über einen `Rechtsklick` auf `Sites`, kann eine neue Webanwendung erstellt werden, auf die per Visual Studio Enterprise 2015 direkt zugegriffen werden kann.

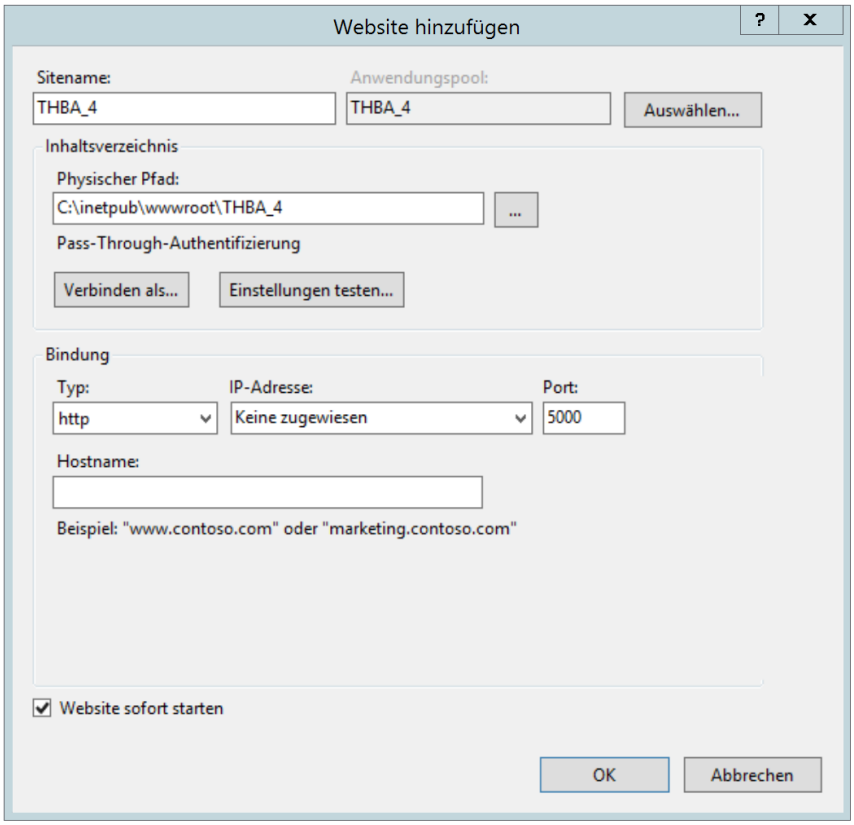


Abbildung 8.14.7 Erstellung einer Website

Unter der Gruppierung `Bindung` lassen sich Protokoll, IP-Adresse und Port festlegen über die die Webanwendung erreichbar sein wird. Sollte als Protokoll `https` ausgewählt werden, ist es notwendig ein SSL-Zertifikat für die Webanwendung zu hinterlegen. In Visual Studio Enterprise 2015 wird über Web Deploy direkt auf die erstellte Website zugegriffen. Im Menü `Erstellen` findet sich der Eintrag `veröffentlichen`.

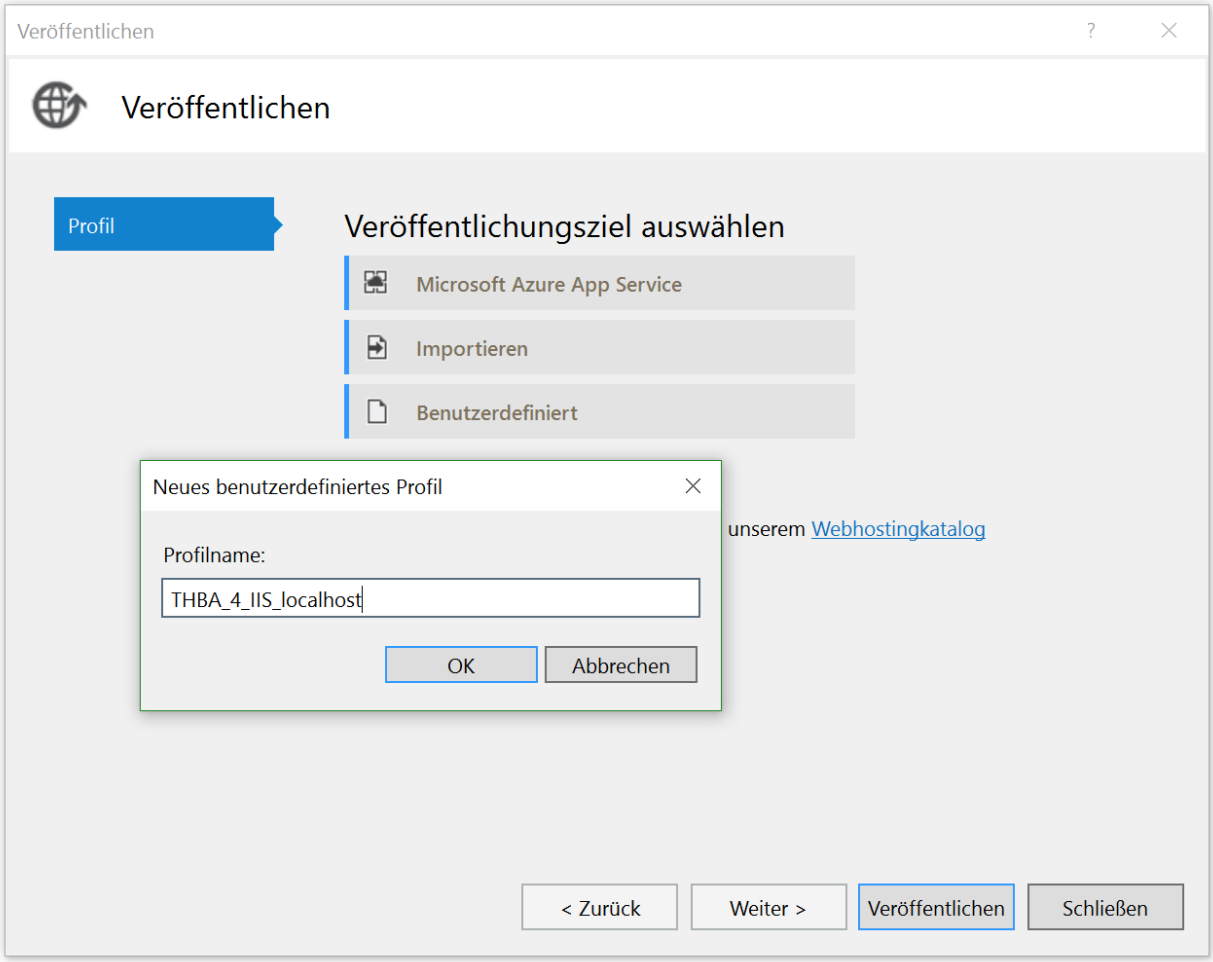


Abbildung 8.14.8 Benutzerdefiniertes Profil erstellen

Für die Nutzung eines Microsoft Internet Information Services (IIS), wird ein benutzerdefiniertes Profil erstellt. Als Veröffentlichungsmethode ist es möglich `Web-Deploy` auszuwählen. Servername und Websitename sind Pflichtangaben. Benutzername und Kennwort sind notwendig, wenn die Website nicht lokal gehostet wird. Es werden administrative Rechte auf dem Rechner benötigt, welcher die Website zur Verfügung stellt. Die Verbindung zum Microsoft Internet Information Services (IIS) sollte überprüft werden.

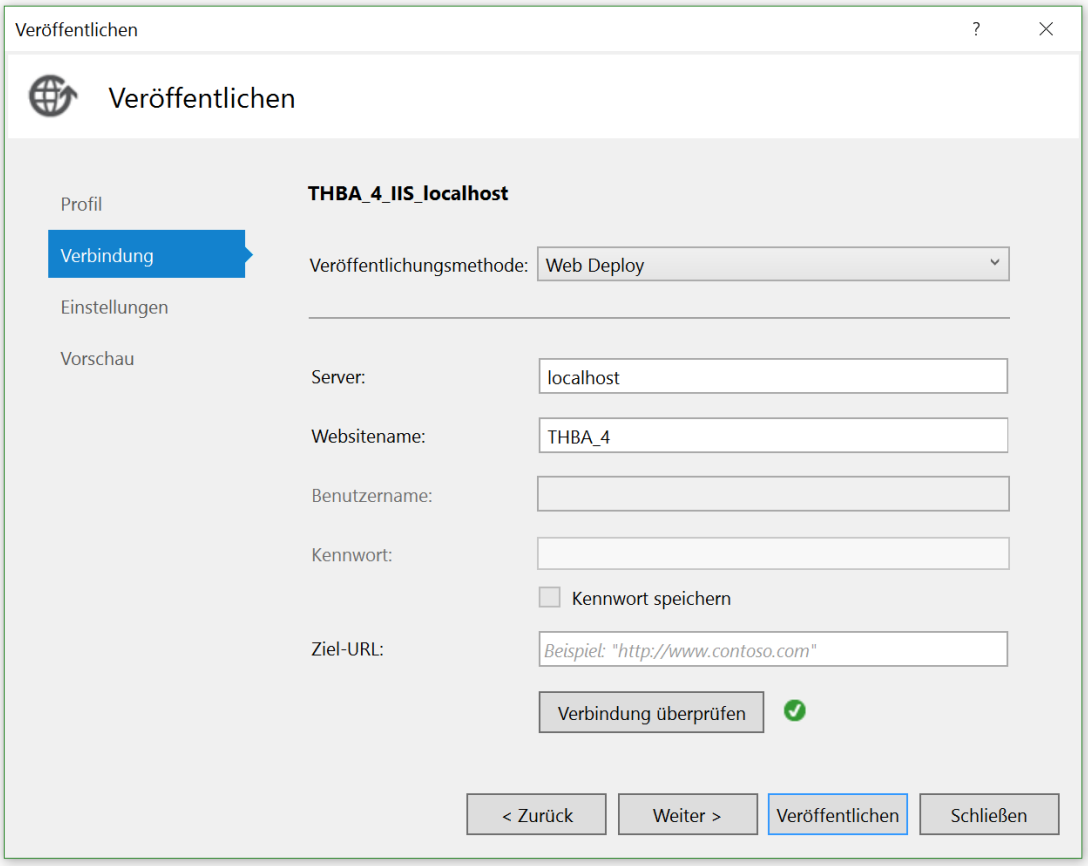


Abbildung 8.14.9 Veröffentlichungsprofil für `Web-Deploy`

Die Konsolenausgabe von Visual Studio Enterpriese 2015 gibt Ausschuss über den Veröffentlichungsvorgang. Fehler und Warnungen werden ausgegeben.

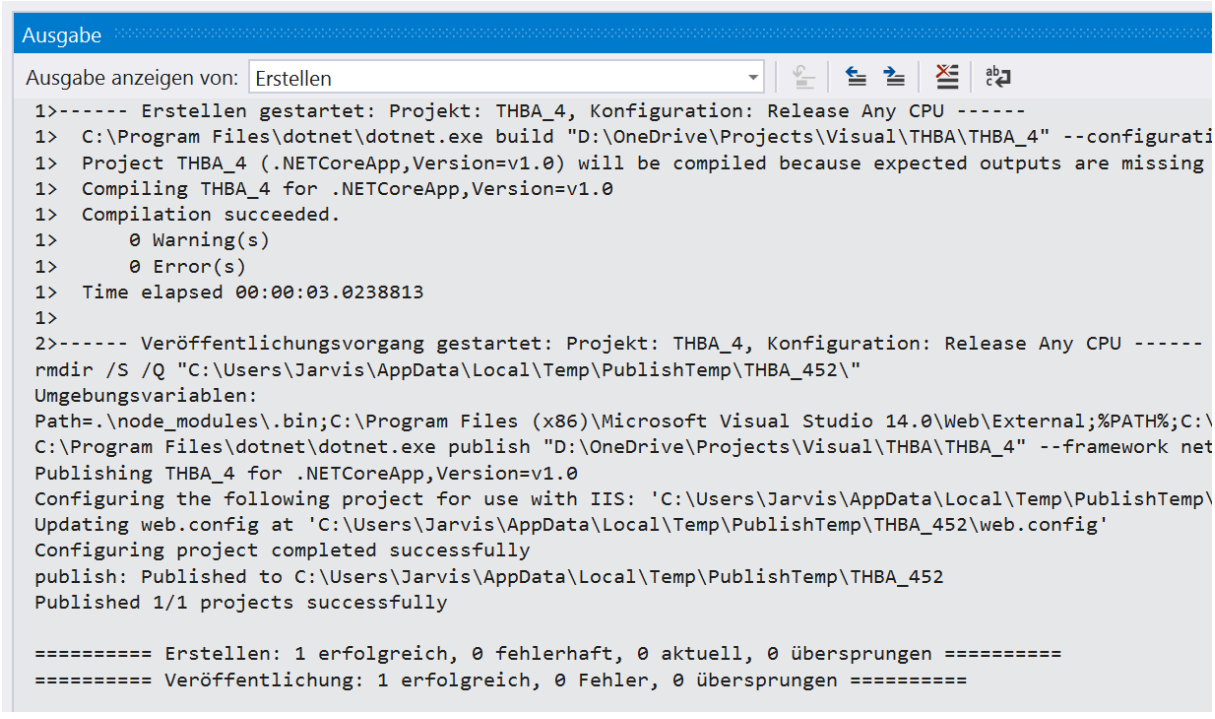


Abbildung 8.14.10 Konsolenausgabe für Web-Deploy

Wurde die Veröffentlichung erfolgreich abgeschlossen, ist die ASP.NET 5 MVC 6 RC1 Web-App über den Browser erreichbar. Die am Microsoft Internet Information Services (IIS) für die Website hinterlegten Einstellungen, entsprechen denen des Aufrufes.

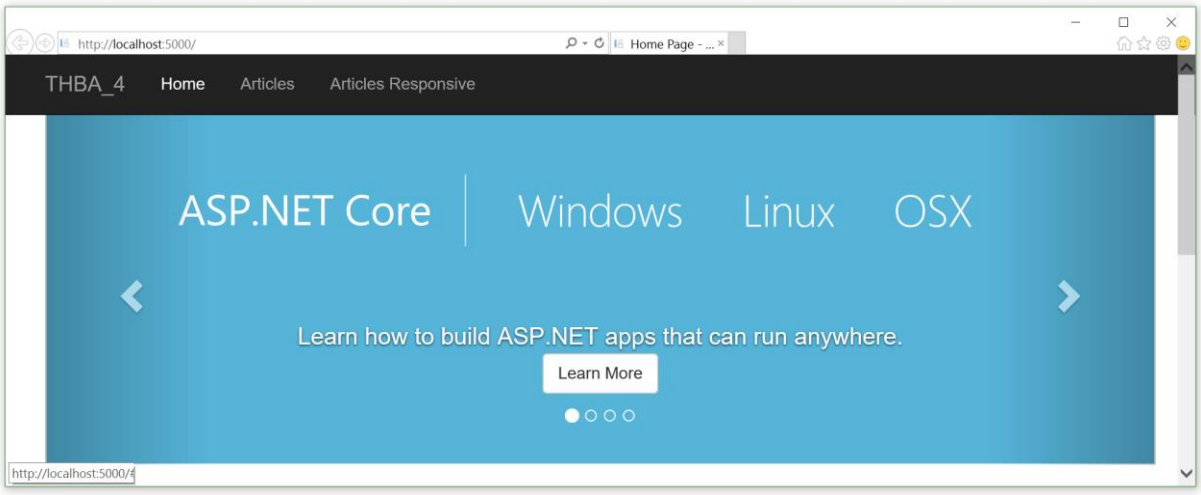


Abbildung 8.14.11 Browserausgabe gehosteter Web-App auf IIS

Die Veröffentlichung der ASP.NET 5 MVC 6 RC1 Webanwendung auf einem Microsoft Internet Information Services (IIS) ist somit abgeschlossen.

8.14.3. Microsoft Cloudplattform Azure

Als dritte Möglichkeit ASP.NET 5 MVC 6 RC1 Web-Apps zu veröffentlichen bietet sich die Microsoft Cloudplattform Azure an. Bei der Microsoft Cloudplattform Azure handelt es sich um eine Sammlung von einzelnen Clouddienste. Virtuelle Maschinen mit unterschiedlichen Betriebssystemen können implementiert werden. Datenbank, Linux-Container und Docker-Integration werden von der Microsoft Azure Plattform unterstützt. Mithilfe der App Services werden Anwendungen für mobile Endgeräte und Web-Apps zur Verfügung gestellt.[56] Die App Services werden für die Veröffentlichung von ASP.NET 5 MVC 6 RC1 Webanwendungen verwendet. Auf die App Services kann per Visual Studio Enterprise 2015 zugegriffen werden. Um auf der Microsoft Azure Plattform Webanwendungen hosten zu können, wird ein Microsoft Azure Konto benötigt. Dies kann auf der Microsoft Azure Webseite erstellt werden.

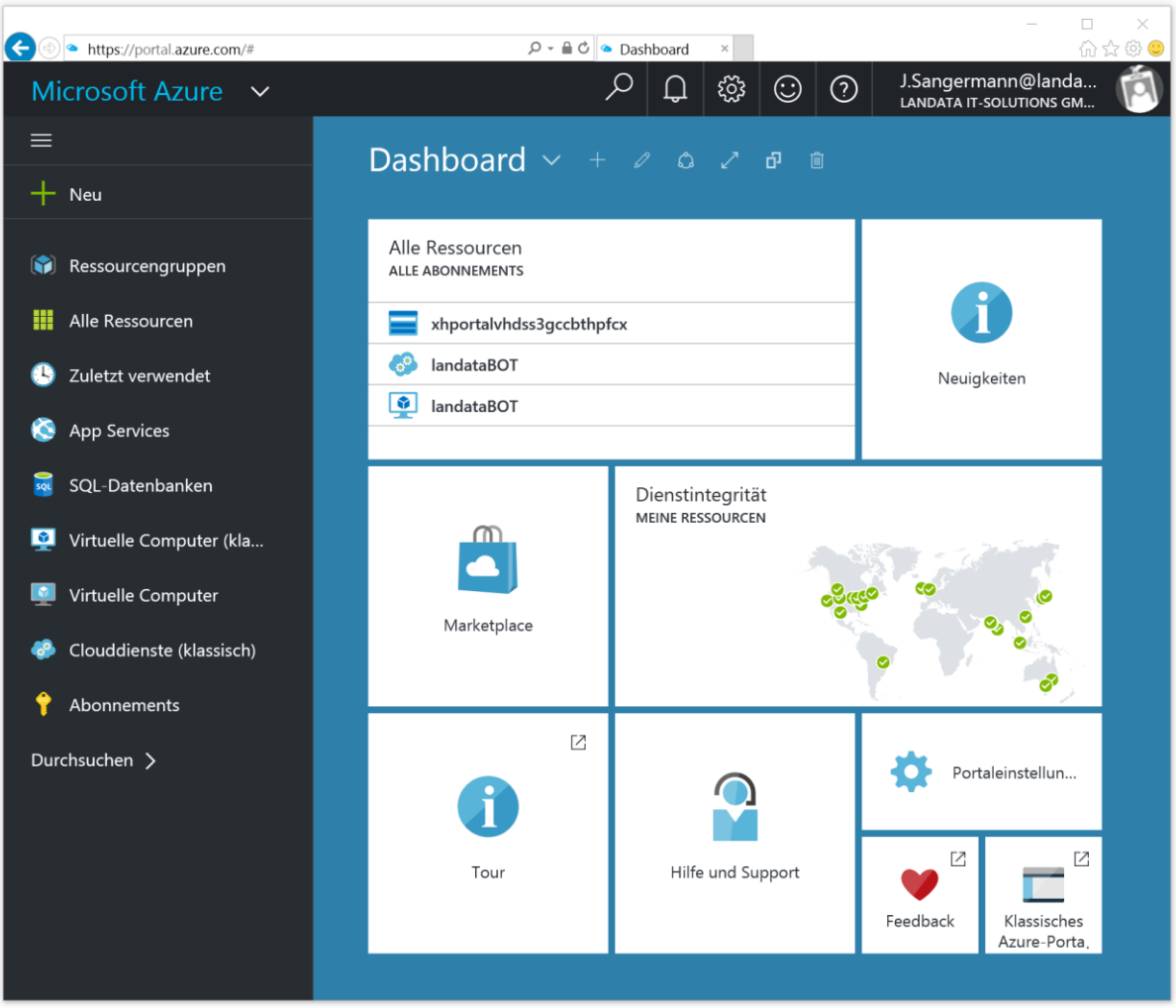


Abbildung 8.14.12 Microsoft Azure Portal Dashboard

Bevor eine Datenbank und ein App Service für eine ASP.NET 5 MVC 6 RC1 Web-App erstellt werden kann, wird eine Ressourcengruppe benötigt. In Ressourcengruppen

[56] Vgl. (Microsoft Azure, 2016), What is Azure

werden zusammenhängende Azure-Dienste zusammengefasst. Beispielsweise besteht eine Webanwendung aus den einzelnen Webseiten und einer Datenbank. Dann würden sich die Datenbank und die Webanwendung in einer Ressourcengruppe befinden. Eine Ressourcengruppe wird über das Menü `Ressourcengruppen` > `Hinzufügen` erstellt.

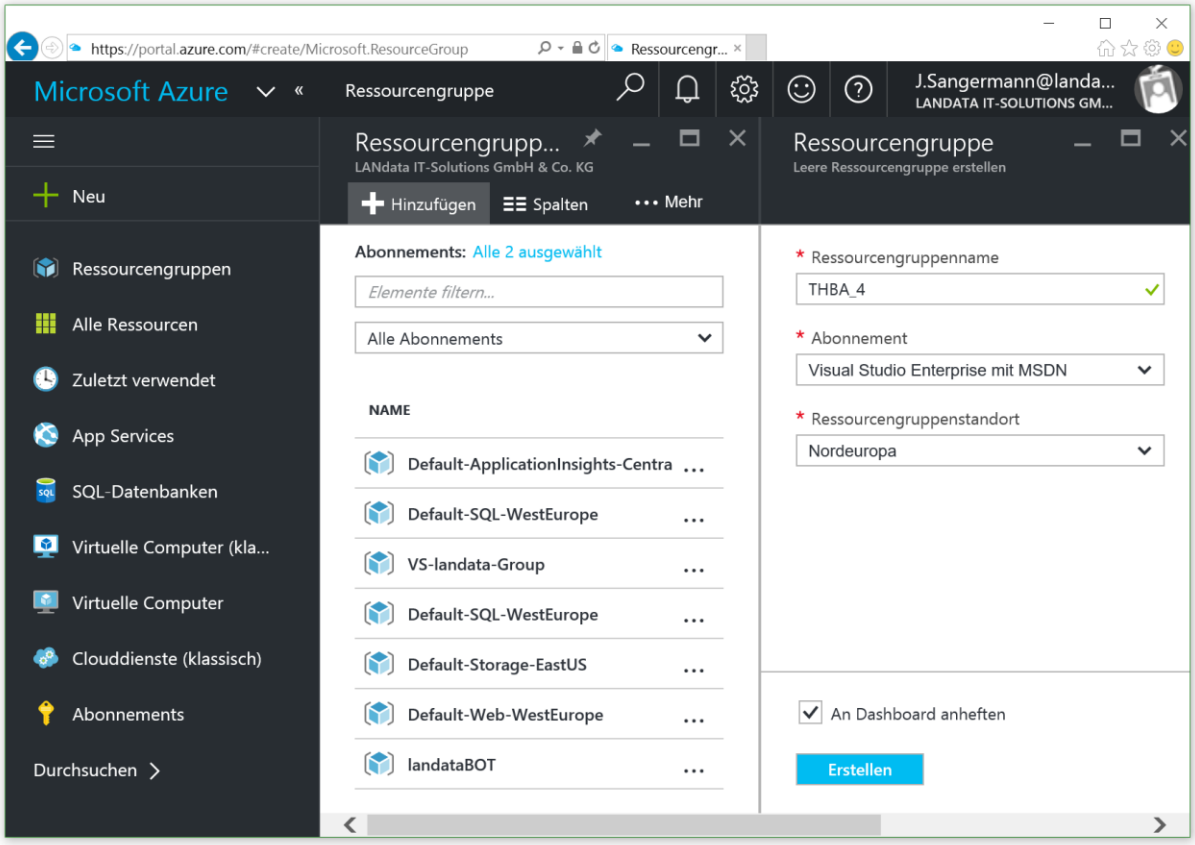


Abbildung 8.14.13 Erstellung einer Ressourcengruppe auf Azure

Da für eine ASP.NET 5 MVC 6 RC1 Web-App eine Datenbank unabdingbar ist, muss eine SQL-Datenbank erstellt werden. Dies kann über das Menü `SQL-Datenbank` > `Hinzufügen` bewerkstelligt werden. Unter dem Eintrag `Ressourcengruppe` kann nun die erstellte Ressourcengruppe für die Anwendung ausgewählt werden. Unter `Server` wird der SQL-Server für die Datenbank konfiguriert. Auf diesen Einstellungen basiert die Verbindungszeichenfolge (`connectionstring`) der ASP.NET 5 MVC 6 RC1 Web-App. Bei `Tarif` kann die Leistung des Datenbankservers eingestellt werden. Es gilt, umso leistungsstärker, desto teurer ist der Server. Auf die einzelnen Profile der Server, die hierbei zur Auswahl stehen, wird hier nicht weiter eingegangen. Beschreibungen zu Preis und Leistung der Server sind bei den einzelnen Profilen gelistet.

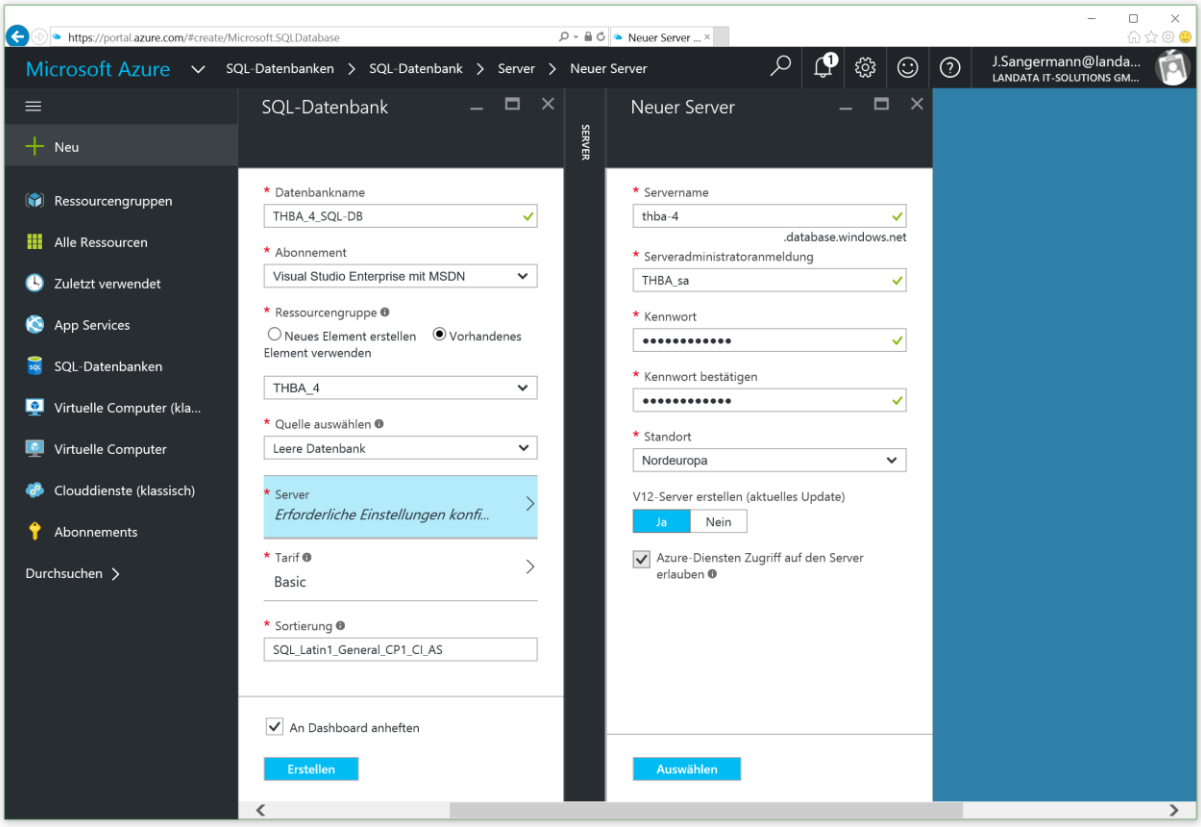


Abbildung 8.14.14 Erstellen einer Datenbank auf Azure

Die App Services hosten die ASP.NET 5 MVC 6 RC1 Webanwendung. Über den dazugehörigen Menüpunkt lassen sich die App Services konfigurieren und einer Ressourcengruppe zuordnen.

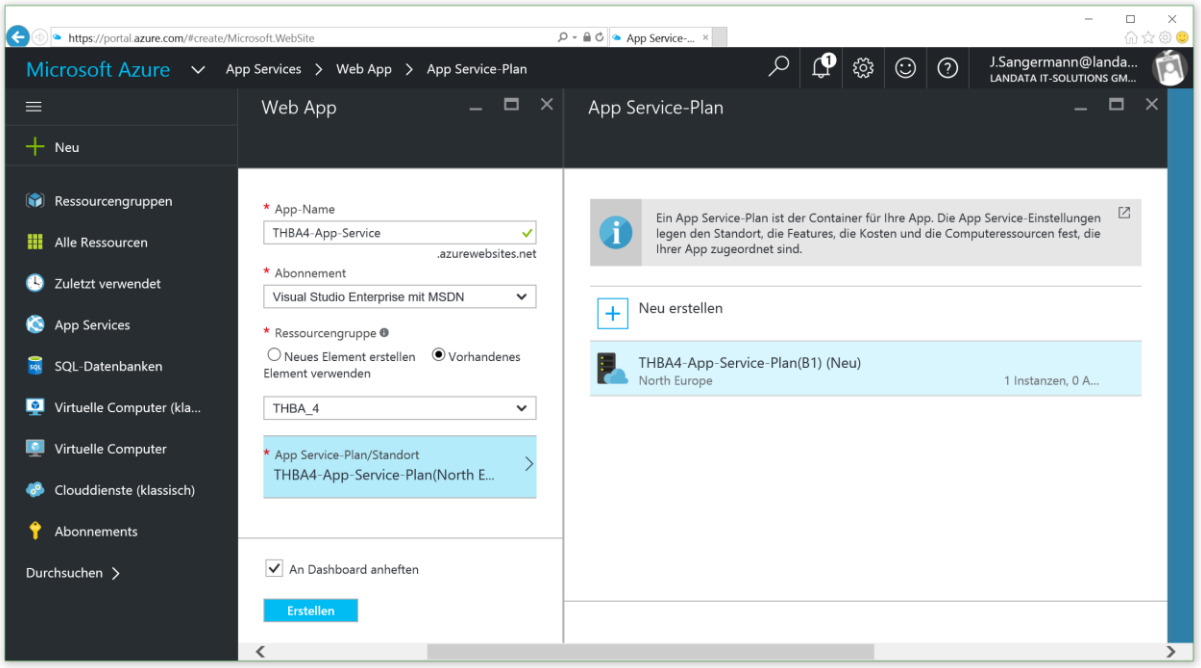


Abbildung 8.14.15 Konfiguration eines App Services auf Azure

Nach der Erstellung des App Services ist es möglich, dass eine ASP.NET 5 MVC 6 RC1 Web-App aus Visual Studio Enterprise 2015 auf der Microsoft Cloud Plattform Azure veröffentlicht werden kann. Analog, wie bei der Veröffentlichung einer Webanwendung auf einem Microsoft Internet Information Services (IIS) (siehe Kapitel 8.14.2 Microsoft Internet Information Services (IIS), Seite 75), muss für das Veröffentlichen ein Profil definiert werden (Menüpunkt `Erstellen` > `veröffentlichen`).

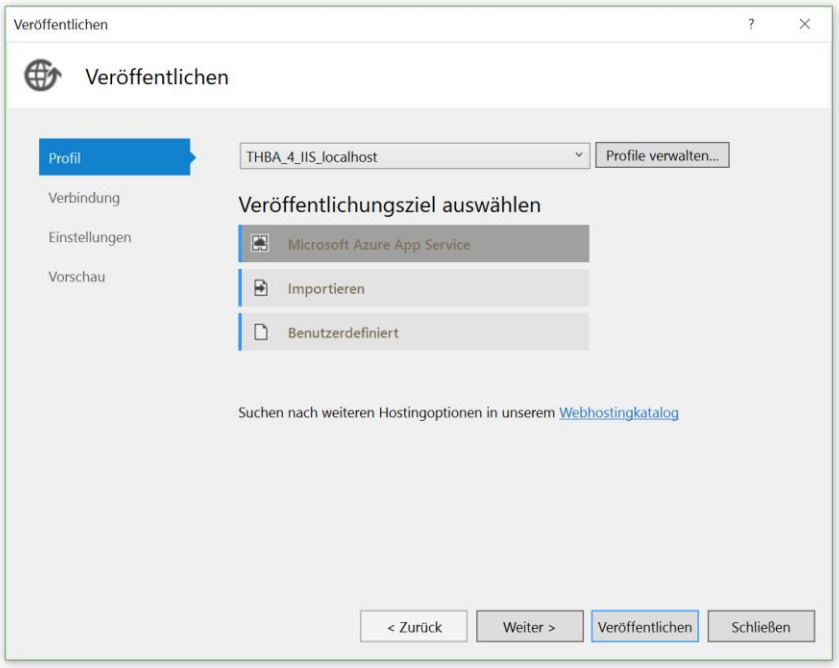


Abbildung 8.14.16 Profil für Microsoft Azure App Service erstellen

Über das Microsoft Azure Konto werden die konfigurierten Ressourcen verwendet.

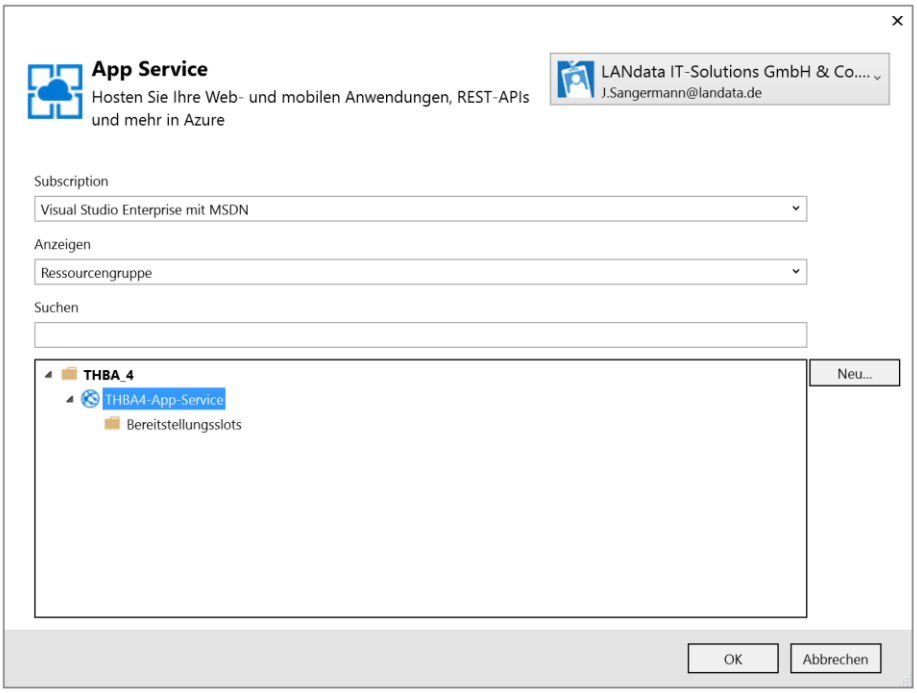


Abbildung 8.14.17 Verbindung zum konfigurierten App Service auf Azure

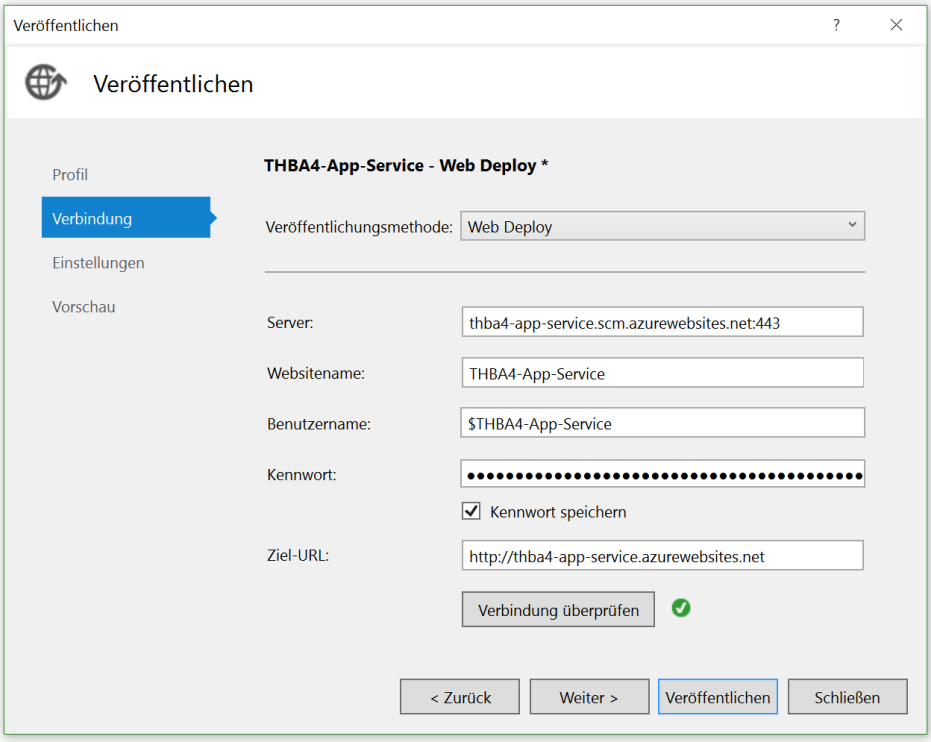


Abbildung 8.14.18 Veröffentlichungsprofil für Azure

Das Profil zum Veröffentlichen der ASP.NET 5 MVC 6 RC1 Web-App wird automatisch von der Microsoft Cloudplattform Azure abgerufen. Unter `Einstellungen` wird die SQL-Datenbank ausgewählt. Die Verbindung sollte abschließend überprüft werden.

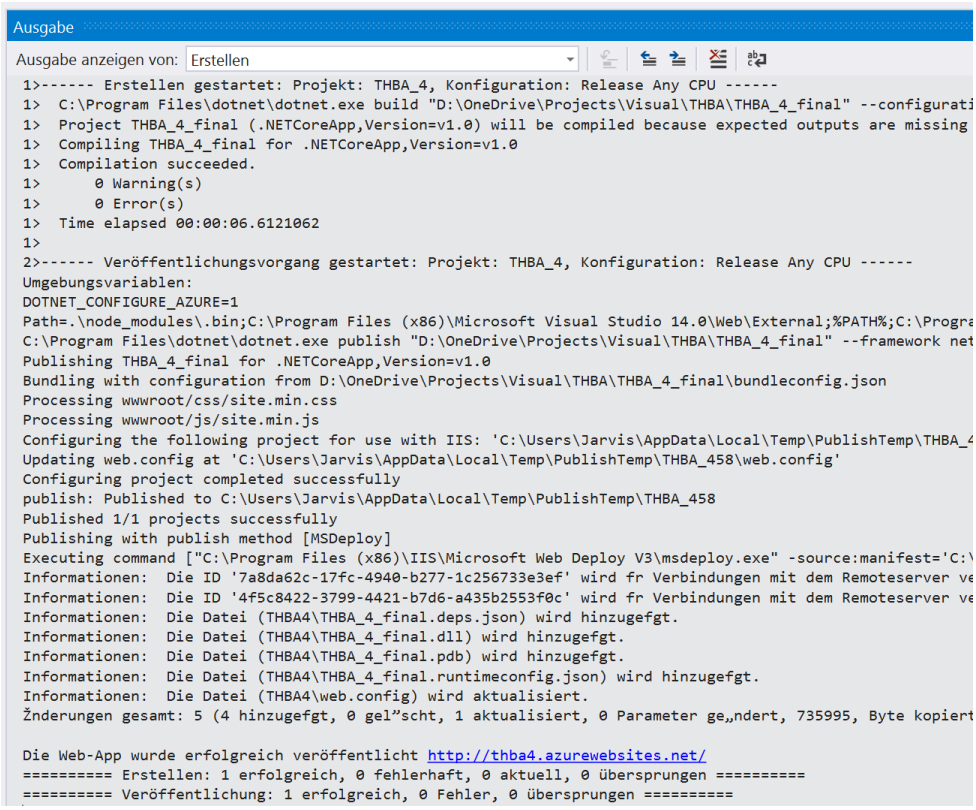


Abbildung 8.14.19 Konsolenausgabe für Azure-Deployment

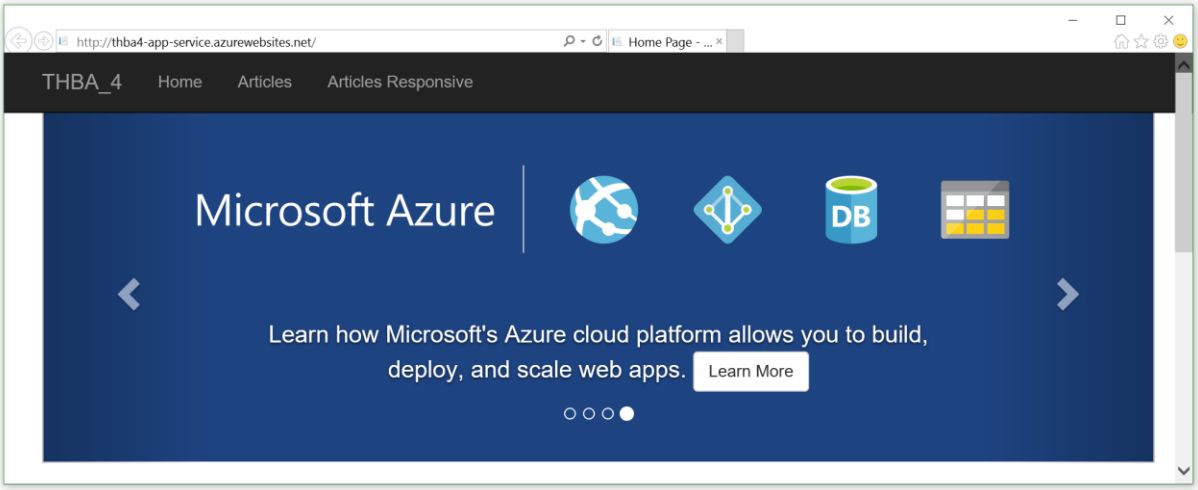


Abbildung 8.14.20 Browserausgabe veröffentlichter Web-App auf Azure

Die ASP.NET 5 MVC 6 RC1 Webanwendung ist damit auf der Microsoft Azure Cloudplattform veröffentlicht und erreichbar.

Der Prozess des Entwickelns und Veröffentlichens einer ASP.NET 5 MVC 6 RC1 Web-App wurde gezeigt und erläutert.

9. Fazit

In diesem Kapitel wird beschrieben inwieweit die Ziele aus der Einleitung dieser Arbeit erreicht worden sind. Zum Ende wird ein Ausblick formuliert. Ergänzend werden offen Fragen dargelegt.

9.1. Zielerreichung

Ziel der Arbeit ist es, dass der Prozess der Erstellung und Veröffentlichung einer ASP.NET 5 MVC 6 RC1 Web-App gezeigt und erläutert wird. Dieses Ziel ist erfüllt worden, auch wenn bei der Erstellung dieser Arbeit Punkte aufgetreten sind, welche die Arbeit erweitern würden (siehe Kapitel 9.3 Offene Fragen, Seite 86). Für Anfänger und professionelle Entwickler ist diese Arbeit geeignet, um sich in das Thema ASP.NET 5 MVC 6 RC1 einzuarbeiten. Jedoch ist zu erwähnen, dass die Arbeit die fundamentalen Aspekte zeigt, welche für die Erstellung einer ASP.NET 5 MVC 6 RC1 Web-App notwendig sind. Die Arbeit gibt einen Einstieg in ASP.NET 5 MVC 6 RC1.

9.2. Ausblick

Während der Erstellung dieser Arbeit, ist die Version von ASP.NET 5 MVC 6 RC1 offiziell in ASP.NET Core 1.0 Preview 1 von seitens Microsoft umbenannt worden. Geplant war dies bereits seit Mitte Januar 2016. [57] Die aktuellste Version lautet ASP.NET Core 1.0 Preview 2 und ist am 27. Juni 2016 erschienen. [58] Die Änderungen in Version ASP:NET Core 1.0 Preview 2 sind in der ASP.NET Dokumentation zu finden. [59] Es ist zu empfehlen, dass die aktuellste Version der Dokumentation zur Erstellung einer ASP.NET 5 MVC 6 RC1 Web-App genutzt wird. Diese ist ebenfalls in der ASP.NET Dokumentation zu finden und wird täglich aktualisiert. [60]

9.3. Offene Fragen

Da sich diese Arbeit auf die Version ASP.NET 5 MVC 6 RC1 und der dazugehörigen Version der Dokumentation bezieht, sind einzelne Punkte dieses Frameworks noch nicht veröffentlicht. Beispielsweise ist die offizielle Version der Razor-Referenz noch nicht erschienen. Auch in der Version Preview 2 fehlt die Referenz. In der finalen Version wird die Razor-Referenz vorhanden sein. Auf das Thema Optimierungen einer ASP.NET 5 MVC 6 RC1 Webanwendung wurde in dieser Arbeit nicht eingegangen, da dies ein weiterführendes Thema für professionelle Entwickler darstellt. Optimierungen wie Gulp.js[61], Grunt.js[62] und Knockout.js[63] werden unterstützt. Das Thema Leistung einer ASP.NET 5 MVC 6 RC1 Web-App wird in der Rubrik Performance der

---

[57] Vgl. (Microsoft Developer Network, 2016), An Update on ASP.NET Core and .NET Core
[58] Vgl. (Microsoft Developer Network, 2016), Announcing .NET Core 1.0
[59] Siehe (Microsoft ASP.NET, 2016), ASP.NET Core Documentation RC2
[60] Siehe (Microsoft ASP.NET, 2016), ASP.NET Core Documentation 1.0
[61] Siehe (Gulp.js, 2016), Durchstarten mit Gulp.js - Websites optimieren, Arbeitsabläufe automatisieren
[62] Siehe (Grunt.js, 2016), Grunt - The JavaScript Task Runner
[63] Siehe (Knockout.js, kein Datum), Introduction

ASP.NET Dokumentation[64] beschrieben. Die Rubrik ist nur teilweise veröffentlicht. Es ist möglich, dass ältere ASP.NET MVC Webanwendung auf ASP.NET 5 MVC 6 RC1 migriert werden. Da dies ein weiterführendes Thema darstellt, wurde es in dieser Arbeit nicht behandelt. Es sei vollständigkeitshalber erwähnt. Bei ASP.NET Web Api[65] handelt es sich um ein Framework, mit dem http-Dienste, wie RESTful-Webservices erstellt werden können. Dies ist vor allem für die Entwickler interessant, die mobile Apps programmieren und eine Anbindung an eine Webanwendung umsetzen möchten. In dieser Arbeit wurde das Entity-Framework[66] für Datenbanken verwendet. Auf die Funktionsweise des Frameworks wurde nicht weiter eingegangen, da dies ein eigenes Thema darstellt. Die Dokumentation ist auf einer separaten Microsoft Webseite zu finden. Bei der Microsoft Azure handelt es sich um eine Cloudplattform, die bei weitem mehr Möglichkeiten bietet, als das Veröffentlichen von Webanwendungen. Mit der Microsoft Azure Cloudplattform sollten sich Entwickler im Microsoftumfeld befassen. In dieser Arbeit wurde kurz auf diese Plattform eingegangen.

[64] Siehe (Microsoft ASP.NET, 2016), ASP.NET Core Documentation 1.0
[65] Siehe (Microsoft ASP.NET, 2016), Learn About ASP.NET Web API
[66] Siehe (Microsoft Entity Framework, 2016), Entity Framework Core

10. Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Olpe, 04.08.2016

John Marc Sangermann
(Matrikelnummer 11070719)

11. Quellcodeverzeichnis

12. Literaturverzeichnis

Bootstrap. (2016). Von About: http://getbootstrap.com/about/ abgerufen

Bower. (2016). Von Bower - A package manager for the web: https://bower.io/ abgerufen

ConnectionStrings.com. (2014). Von SQL Server connection strings: https://www.connectionstrings.com/sql-server/ abgerufen

Docker. (2016). Von What is Docker?: https://www.docker.com/what-docker abgerufen

Grunt.js. (2016). Von Grunt - The JavaScript Task Runner: http://gruntjs.com/ abgerufen

Gulp.js. (27. Juli 2016). Von Durchstarten mit Gulp.js - Websites optimieren, Arbeitsabläufe automatisieren: http://magazin.phlow.de/webdesign/gulp/ abgerufen

Habib, M. (15. Oktober 2012). *Code Project*. Von What is ViewData, ViewBag and TempData? – MVC Options for Passing Data Between Current and Subsequent Request: http://www.codeproject.com/Articles/476967/What-is-ViewData-ViewBag-and-TempData-MVC-Option abgerufen

IT Wissen. (kein Datum). Von URL (uniform resource locator): http://www.itwissen.info/definition/lexikon/uniform-resource-locator-URL.html abgerufen

jQuery. (2016). Von jQuery API: http://api.jquery.com/ abgerufen

Knockout.js. (kein Datum). Von Introduction: http://knockoutjs.com/documentation/introduction.html abgerufen

Marcotte, E. (25. Mai 2010). *Alistapart*. Von Responsive Web Design: http://alistapart.com/article/responsive-web-design abgerufen

Microsoft ASP.NET. (7. Februar 2014). Von Introduction to ASP.NET Web Programming Using the Razor Syntax (C#): http://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-c abgerufen

Microsoft ASP.NET. (2016). Von Role based Authorization: https://docs.asp.net/en/latest/security/authorization/roles.html abgerufen

Microsoft ASP.NET. (2016). Von ASP.NET Core Documentation RC1: https://docs.asp.net/en/1.0.0-rc1/ abgerufen

Microsoft ASP.NET. (2016). Von ASP.NET Core Documentation RC2: https://docs.asp.net/en/1.0.0-rc2/ abgerufen

Microsoft ASP.NET. (2016). Von ASP.NET Core Documentation 1.0: https://docs.asp.net/en/latest/ abgerufen

Microsoft ASP.NET. (2016). Von Learn About ASP.NET Web API: http://www.asp.net/web-api abgerufen

Microsoft ASP.NET. (2016). Von Introduction to Identity: https://docs.asp.net/en/1.0.0-rc1/security/authentication/identity.html abgerufen

Microsoft ASP.NET. (2016). Von Servers: https://docs.asp.net/en/1.0.0-rc1/fundamentals/servers.html#kestrel abgerufen

Microsoft Azure. (2016). Von Ihre App. Ihr Framework. Ihre Plattform. Platz für alles.: https://azure.microsoft.com/de-de/overview/what-is-azure/ abgerufen

Microsoft Developer Network. (kein Datum). Von Richtlinien für die ASP.NET-Webkonfiguration: https://msdn.microsoft.com/de-de/library/ff400235(VS.100).aspx abgerufen

Microsoft Developer Network. (2010). Von ASP.NET MVC Overview: https://msdn.microsoft.com/de-de/library/dd381412(v=vs.108).aspx abgerufen

Microsoft Developer Network. (2015). Von Visual Studio-IDE: https://msdn.microsoft.com/de-de/library/dn762121.aspx abgerufen

Microsoft Developer Network. (2015). Von C#: https://msdn.microsoft.com/de-de/library/kx37x362.aspx abgerufen

Microsoft Developer Network. (April 2016). Von .NET-Grundlagen: Protokollieren mit .NET Core: https://msdn.microsoft.com/de-de/magazine/mt694089.aspx abgerufen

Microsoft Developer Network. (Februar 2016). Von Grundlagen zum Komponententest: https://msdn.microsoft.com/de-de/library/hh694602.aspx abgerufen

Microsoft Developer Network. (Juli 2016). Von Verwenden von Eigenschaften (C#-Programmierhandbuch): https://msdn.microsoft.com/de-de/library/w86s7x04.aspx abgerufen

Microsoft Developer Network. (27. Juni 2016). Von Announcing .NET Core 1.0: https://blogs.msdn.microsoft.com/dotnet/2016/06/27/announcing-net-core-1-0/ abgerufen

Microsoft Developer Network. (27. Juni 2016). Von .NET Web Development and Tools Blog: https://blogs.msdn.microsoft.com/webdev/2016/06/27/announcing-asp-net-core-1-0/ abgerufen

Microsoft Developer Network. (01. Februar 2016). Von An Update on ASP.NET Core and .NET Core: https://blogs.msdn.microsoft.com/webdev/2016/02/01/an-update-on-asp-net-core-and-net-core/ abgerufen

Microsoft Docs. (26. Juli 2016). *Microsoft Documentation*. Von project.json reference: https://docs.microsoft.com/de-de/dotnet/articles/core/tools/project-json abgerufen

Microsoft Entity Framework. (2016). Von Entity Framework Core: https://docs.efproject.net/en/latest/intro.html abgerufen

Microsoft IIS. (kein Datum). Von IIS Overview: http://www.iis.net/overview abgerufen

Microsoft IIS. (12. Dezember 2011). Von Installing and Configuring Web Deploy on IIS 7: http://www.iis.net/learn/install/installing-publishing-technologies/installing-and-configuring-web-deploy abgerufen

NuGet. (2008). Von NuGet Documentation: https://docs.nuget.org/consume/package-manager-console abgerufen

NuGet. (2016). Von NuGet: https://www.nuget.org/ abgerufen

Rasperry Pi. (kein Datum). Von About Us: https://www.raspberrypi.org/about/ abgerufen

Schütze, J. (2008). Von Verteilte Versionsverwaltung: http://www.fh-wedel.de/~si/seminare/ws08/Ausarbeitung/06.vvw/vvw1.htm abgerufen

Tecchannel. (30. August 2001). Von HTTP-Grundlagen - Hypertext Transfer Protocol: http://www.tecchannel.de/a/hypertext-transfer-protocol,401210,6 abgerufen

xUnit.net. (2015). Von About xUnit.net: https://xunit.github.io/ abgerufen